Generación de componentes para la creación y evaluación de poemas musicales

Trabajo de Fin de Grado
Curso 2018–2019

Autor
Carlos Martín Testillano
Mª Luisa Quiroga Fernández-Miranda

Director
Gonzalo Méndez Pozo
Pablo Gervás Gómez-Navarro

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Generación de componentes para la creación y evaluación de poemas musicales

Trabajo de Fin de Grado en Ingeniería Informática
Departamento de Ingeniería de Software e Inteligencia Artificial

Autor
Carlos Martín Testillano
Mª Luisa Quiroga Fernández-Miranda

Director
Gonzalo Méndez Pozo
Pablo Gervás Gómez-Navarro

Lyrics generation and evaluation modules

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

20 de septiembre de 2019

Acknowledgements

First of all, we would like to thank our directors Gonzalo and Pablo for the help they have given us during this year, and for transmitting us their love and interest in art and computational creativity. We would also like to thank them for their availability and willingness to guide and recommend us, without which we would not have been able to finish this project.

We want to give a special mention to our colleague, but above all friend, Guille. He has been with us not only during this year, but since day one in university. We would like to thank him for his work and transmitting us his ability to overcome every obstacle that ever comes before us. This work is also yours.

Finally, we would also like to acknowledge all our families and friends who have been with us along this year, trusting, lifting and pushing us whenever we needed their help.

Resumen

En el campo de la creatividad computacional ha habido varias investigaciones relacionadas con la generación automática de música y de textos literarios. En este trabajo proponemos una solución para la generación automática de canciones, primero escribiendo la música y la letra por separado, y más tarde uniéndolas.

En relación a la generación de música, se ha usado un módulo desarrollado por Guillermo Villeta Torres que será explicado en un futuro TFG. Este módulo genera canciones en un formato MIDI que posteriormente se juntarían con la letra, y también es capaz de modificar la música en función del feedback que recibe tras esta unión.

Para la generación de letras exploramos el uso de cadenas de Markov, que nos ayudan a obtener texto poético. Durante este proceso desarrollamos algunas pruebas para cada oración generada con el fin de llevar a cabo una evaluación sintáctica y un análisis de los sentimientos que genera. Para ambas evaluaciones, se probaron diferentes configuraciones hasta que el resultado obtenido era el más satisfactorio.

Una vez la música y la letra han sido generadas, son enviadas a otro módulo que las junta y evalúa el resultado obtenido. Durante el proceso de evaluación, definimos una serie de pruebas, primero para hace un análisis de los sentimientos de la música, y más tarde para obtener un índice de calidad de la canción resultante. Estas pruebas se llevan a cabo iterativamente a lo largo de toda la música con la letra ya asignada, y más tarde la nota de calidad se normaliza. En función del resultado de este proceso de evaluación, o bien la canción resultante se da como salida, o se manda una serie de parámetros a los módulos generadores para que modifiquen tanto la letra como la música con la intención de mejorar el resultado.

Palabras clave

Creatividad computacional, modelo de Markov, música, letra, evaluación, análisis de sentimientos, composición.

Abstract

In the computational creativity field there have been many researches related to the automatic generation of music and forms of literature. In this work we propose a solution to the automatic generation of songs, first writing the music and the lyrics independently, and later joining them.

Regarding music generation, a module developed by Guillermo Villeta Torres that would be explained in a later TFG, has been used. This module generates a song in MIDI format that would be then joined to the lyrics, and is also capable of modifying the music based on the feedback it receives after this merging.

As for the generation of lyrics, we explored the use of Markov Chains which helped us obtain a poetic text. During this process, we developed some tests for each generated sentence to carry a syntactic and a sentimental evaluation. For both evaluations, different configurations were tried until the result obtained was the most satisfactory one.

When both music and lyrics are ready, they are received by another module that put them together and evaluated the outcome. During this evaluation process, we came up with a series of tests first to check the sentiment analysis of the music, and then to obtain a quality index of the resulting song. These tests are performed iteratively through the music with the lyrics already assigned, and later the quality score is normalized. Depending on the outcome of this evaluation process, either the result is given as an output or a series of parameters are sent to the generator modules as feedback for them to modify the lyrics and music with the intention of improving the result.

Keywords

Computational creativity, Markov model, music, lyrics, evaluation, sentiment analysis, song-writing.

Contents

List of figures

List of tables

Chapter 1

Introducción

"There, in the chords and melodies, is everything I want to say. The words just jolly it along. It's always been my way of expressing what, for me, is inexpressible by any other means."
— David Bowie

1.1. Motivación

Desde el principio de los tiempos, los humanos han usado el arte de diferentes formas para transmitir emociones, sentimientos o devoción. Entre estás expresiones del arte podemos encontrar pinturas, esculturas, música o literatura, y todas ellas han evolucionado con el tiempo hasta nuestros días.

En relación a la música, las composiciones escritas más antiguas datan de la Antigua Grecia (siglo 128 A.C.) o incluso siglos anteriores. Comparando una composición musical de esta época con una más actual es fácil percibir las diferencias, no sólo en el lenguaje, sino también en las representaciones utilizadas para trasmitir la melodía o los instrumentos con la que debe ser tocada. Lo mismo ocurre con la poesía, que es la manera más artística de expresar el lenguaje al basarse en la estética, ya que ha visto una evolución similar en forma y significado.

No obstante, esta evolución no se percibe solamente en la manera en la que la música o poesía son representadas o en qué transmite, sino también en cómo se crean. Desde el comienzo han sido las personas solas las que han compuesto una melodía y escrito un poema, pero no fue hasta la generalización de los ordenadores que una figura externa ayudó a los humanos a desarrollar estas obras de arte. Los ordenadores no sólo ayudan a las personas en sus tareas, sino que muchos investigadores han estudiado diferentes maneras en las que puedan llevar a cabo acciones propias de los seres humanos, lo que recibe el nombre de **Creatividad Computacional**.

La creatividad computacional es un campo de estudio muy extenso en el que los investigadores intentan emular la forma de actuar de las personas en una de sus cualidades más humanas como es la creatividad. Principalmente, estos estudiosos se centran en desarrollar programas informáticos y algoritmos que emulan el comportamiento de una persona cuando crea algo que denota imaginación.

Como punto de estudio, se han tratado muchos enfoques para la composición de música o generación de poesía, y también la creación de canciones completas con música y letra (que es esencialmente un poema distribuido a lo largo de la melodía). Estos estudios nos servirán como punto de partida para nuestro proyecto, ya que pensamos que podemos contribuir en este campo diseñando y desarrollando una solución en este área de la creatividad computacional que todavía no ha sido estudiado a fondo.

1.2. Objetivo

El principal objetivo de este proyecto es diseñar y desarrollar una solución para la generación automática de canciones, componiendo la música y escribiendo la letra por separado y luego uniéndolas.

Para conseguir esto, investigaremos en el campo de la creatividad computacional en busca de técnicas y enfoques que se puedan tomar. Durante esta investigación nos centraremos en la generación de poesía para escribir la letra ya que, si se quita la música de una canción, la parte restante se puede considerar un poema.

Para que la solución sea lo más sensata posible, tendremos que profundizar nuestro conocimiento en materia de escritura de poesía, composición de música y creación de canciones. Adicionalmente, tendremos que estudiar las diferentes tecnologías disponibles que nos puedan ayudar a desarrollar nuestra solución.

Nuestro propósito es obtener un sistema que genere una canción en función de unos datos de entrada. Con esta entrada, la música y la letra se generará por separado en módulos independientes. Una vez estos componentes obtengan un resultado válido, lo enviarán a otro módulo que se hará cargo de unirlos. Durante este proceso de combinar la música con la letra, una serie de pruebas se llevarán a cabo para determinar la calidad del resultado propuesto y, en caso de que no sea considerado los suficientemente satisfactorio, este módulo mandará un *feedback* a los dos generadores. Esta información que los componentes recibirían se usará para modificar la música y letra propuestas antes de que se vuelva a intentar la combinación de ambas.

El diseño de la solución se hace teniendo en cuenta futuras evoluciones y modificaciones, tanto de los desarrolladores como de investigadores ajenos. Desarrollaremos la solución incrementalmente, primero trabajando con ejemplos pequeños y controlados, y más tarde usando una fuente de entrada más compleja para los procesos de generación y pruebas de la unión y módulo evaluador.

Por último, nos gustaría contribuir al campo de la creatividad computacional mostrando nuestra perspectiva y proponiendo una solución en un área que aún no se ha estudiado a fondo. Ha habido muchos enfoques con un objetivo similar en mente, pero todos han sido usando partes ya existentes, y queremos mostrar uno en el que se generen canciones desde cero.

En este documento, solo las partes de generación de letras y unión y evaluación serán descritas, ya que el módulo de generación de música se explicará en un proyecto complementario futuro escrito por Guillermo Villeta Torres.

Chapter 1

Introduction

1.1. Background

Since the beginning of time, humans have used different forms of art to transmit emotions, feelings or devotion. Among these forms of art we can find paintings, sculptures, music or literature, and all of them have evolved with time until our days.

Regarding music, the oldest written pieces known come from the Ancient Greece (c.128 BCE) or even centuries earlier. Having side by side a music composition from this era and a more modern one, it is easy to notice the differences, not only in language, but in the representations used for transmitting the melody and the instruments it is meant to be played with. The same happens with poetry, which is the most artistic way of language as it relies on aesthetics, as it has seen the same evolution with time in form and meaning.

However, evolution has not been made just in the way music or poetry are represented and what they transmit, but also in how they are created. Since the start, it has been humans alone that have composed a melody and written a poem, but it was not until the generalization of computers that some external figure helped a person develop these forms of art. Computers not only helped people in their tasks, but many researchers studied different ways in which they could perform the tasks humans were performing, which is called **Computational Creativity**.

Computational creativity is a very wide field of study in which researchers try to mimic the actions of a person in one of its most humanized capabilities as is creativity. Mainly, researches focus on developing computer programs and algorithms that emulate the behaviour of a human when creating something that shows its imaginativeness.

As our focus point of study, many approaches have been taken for the composition of music or generation of poetry, as well as some of them regarding the creation of entire songs with both music and lyrics (which are essentially poems distributed along the melody). These approaches will serve as a starting point for our project, but we believe we can contribute to this field by designing and developing a solution for this computational creativity area that has not yet been deeply studied.

1.2. Objective

The main objective of this project is to design and develop a solution to the automatic generation of songs, composing the music and writing the lyrics and putting them together.

To achieve this, we will investigate into the field of computational creativity in search of techniques and approaches that can be taken. In this research we will focus on poetry generation for the writing of the lyrics, as usually, if the music is taken off of the song, the remaining part can be considered a poem.

In order for the solution to be sensible enough, we will have to deepen our knowledge in the matter of poetry writing, music composing and song-writing altogether. Additionally, we will have to study all the available technologies that would help us develop this solution.

Our purpose is to obtain a system that generates a song given a set of input data. With this input, the music and the lyrics would be generated by two independent modules. Once these components obtain a valid result, they send it to another module that would be in charge of merging both generated outputs. During this process of combining the music with the lyrics, a set of tests would be carried in order to determine the quality of the proposed result and, in the case it is not considered good enough, this module would send feedback to the two generators. This information that the composers would receive would be used to modify the proposed music and lyrics before the combination is tried again.

The design of this solution is made having in mind future improvements and modifications, both by the developers and external researchers. We will develop the solution incrementally, first working with smaller and controlled samples, and later using more a more complex corpus for the generation process and testing of the merging and evaluation module.

Finally, we would like to contribute to the computational creativity field by showing our perspective and proposing a solution in an area that has not yet been thoroughly studied. There have been some approaches with a similar objective in mind, but all were using some existing parts, and we want to show an approach of generating songs from scratch.

In this document only the lyrics generation and merge and evaluation parts will be described, as the music generator module will be explained in a future complementary project written by Guillermo Villeta Torres.

Chapter 2

State of the Art

In this chapter we will talk about the previous work that has been done in the computational creativity field regarding the composition of poetry. We will explain the different approaches taken by various researchers in the past whose purpose was to develop an application capable of writing poetry. We will also consider the composition of music, although that part is not described in this document, but it will be necessary for the merging of the lyrics with the music. Finally, we will present the diverse methods used in the different researches to evaluate the quality of the results obtained.

2.1. Theoretical concepts

In this section we will introduce various theoretical concepts regarding both music and poetry (as the lyrics are considered a poem) that need to be explained in order to understand why the developed solution was designed this way. This explanation is addressed to any reader with a low level of theoretical knowledge of music or poetry but will also help identify the main focus points of the development from a musical and poetical perspective.

2.1.1. Musical concepts

Although music theory is very complex, we have identified some key points of a music composition that would help an inexperienced reader understand a basic music sheet.

2.1.1.1. Pitch

According to the Oxford Dictionary of English, the pitch (also named note) is

> "the quality of a sound governed by the rate of vibrations producing it; the de-

gree of highness or lowness of a tone"footnote[1]`https://en.oxforddictionaries.com/definition/pitch`.

This means that depending on the rate of vibrations generated to make a sound, this will have one pitch or another.

In music, pitches have names associated to simplify their use. These names are composed by a letter from A to G and an accidental (sharp -#- or flat -b-), although the assignation of the latter depends on the frequency, it is not mandatory.

Internally in the application, pitches are managed using its associated MIDI number. This number is a result of a technical standard[2] that normalized frequencies of sound to make them transferable between different devices. The use of these numbers eases the process of managing pitches and their frequencies, as well as the creation of intermediate and output files in MIDI format and music sheets.

Additionally, it is also important to define the concept of interval, which is determined by the difference in pitch between two sounds. Commonly they are identified as the differences in the diatonic scale (see section 2.1.1.2) and the smallest unit of measurement is the *semitone*. In the following figure (2.1) an example of tonal distance can be seen, where **W** means there is a whole tone (two semitones) between the note and the following one, and **H** a half tone (or semitone).

Figure 2.1: Tonal distances example

Considering the definition of intervals, the tendency is defined as the variation in the consecutive pitches. It can be ascending (figure 2.2), descending (figure 2.3) or flat (figure 2.4) depending on the difference between the frequencies of the notes being contemplated.

2.1.1.2. Scales

Scales (also identified as keys) are defined as an organized sequence of notes (pitches). They are sorted from the lowest to the highest frequency and can start in any pitch. Depending on the pitch the scale starts with, and the difference in the intervals between the contiguous notes, the scale can receive one name or another. For simplicity purposes, the application will only work with diatonic scales, which are built with seven pitches and

[2] `https://en.wikipedia.org/wiki/MIDI`

Figure 2.2: Ascending tendency

Figure 2.3: Descending tendency

Figure 2.4: Flat tendency

thus, seven intervals (five whole tones and two semitones) organized depending on the starting note (see figures 2.1 and 2.5).

Figure 2.5: C scale

Notes have always a name associated on their pitch, but they also have one depending on their position in the scale. These names are: *tonic, supertonic, mediant, subdominant, dominant, submediant and leading tone.* Regardless of the key, they are always arranged in the same order, and therefore they are referred with roman numerals (I for the tonic, VII for the leading tone) and not their note names (see figure 2.6).

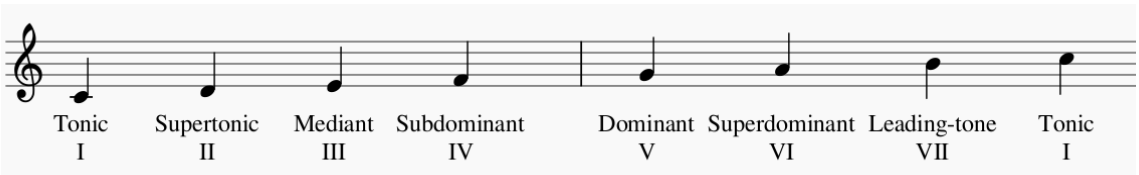

Figure 2.6: C scale position names

2.1.1.3. Modes

Together with the key, every composition has to have been written in a mode, which indicates a way in which a series of notes, or scale, is played. The differences between modes are given by the tonal separation of the notes. Major and minor are the two most common modes, but there are many more modes a melody can be written in. Some of them are so rare they do not have an associated name, but there are others that do. The most used denomination for these modes (apart from major and minor), is the Greek names they took in the past. There are seven, as pitches are in a scale, and are the following: *lydian* (figure 2.7), *ionian* (equivalent to the major mode, figure 2.8), *mixolydian* (figure 2.9), *dorian* (figure 2.10), *aeolian* (equivalent to the minor mode, figure 2.11), *phrygian* (figure 2.12) and *locrian* (figure 2.13).

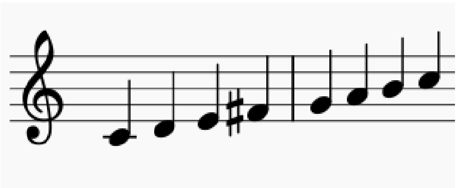

Figure 2.7: Scale in lydian mode

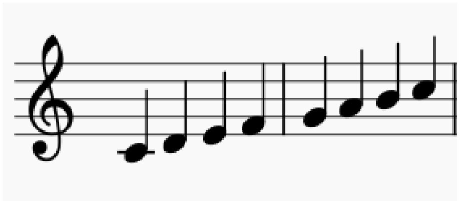

Figure 2.8: Scale in ionian mode

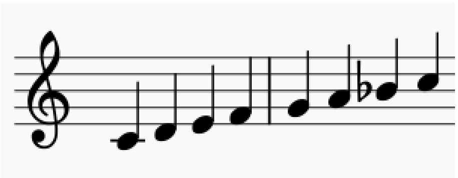

Figure 2.9: Scale in mixolydian mode

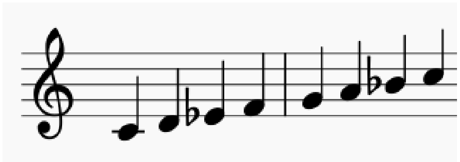

Figure 2.10: Scale in dorian mode

Figure 2.11: Scale in aeolian mode

Figure 2.12: Scale in phrygian mode

Figure 2.13: Descending tendency

2.1.1.4. Note values

Note values indicate the relative duration of notes, which is measured in beats (the basic unit of time in music, also defined as a rhythmic movement). This value can be identified by the design of the note. Apart from sounding notes, there are also rests or silences, which indicate the absence of sound.

There are several note values, but the most important are: *semibreve* (whole note, four beats), *minim* (half note, two beats), *crotchet* (quarter note, one beat), *quaver* (eighth note, half beat) and *semiquaver* (sixteenth note, quarter beat).

Figure 2.14: Semibreve

Figure 2.15: Minim

Figure 2.16: Crotchet

Figure 2.17: Quaver

Figure 2.18: Semiquaver

2.1.1.5. Measures

In a music composition, a measure (or bar) indicates a segment of time corresponding to a specific number of beats. Compositions are thus divided into measures marked by vertical bars across the pentagram, and the number of beats that fit each of these is indicated at the beginning of the composition by the time signature. These two indicators can be seen highlighted in figure 2.19.

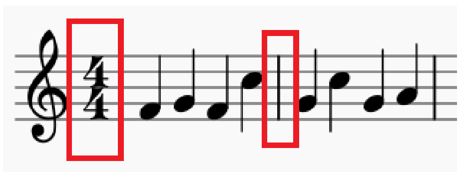

Figure 2.19: Time signature and bar delimitation figures

2.1.1.6. Metronome mark

The metronome mark, or tempo, of a music composition determines the speed at which it is originally supposed to be played. It is more common to see a music sheet with the indication of the metronome mark (figure 2.20) in the cases of classical music, as actual compositions tend to give the musician more freedom of interpretation.

The tempo is measured in bpm (beats per minute), which means that a given figure (crotchet, half note, eighth note…) has to fit that given number of times in a minute. Some concrete metronome marks or intervals of them were assigned a denomination in which they are commonly known (andante, allegro, piano, moderato…), but there are infinite.

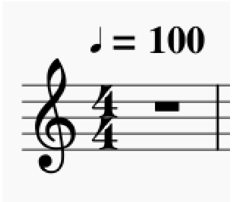

Figure 2.20: Metronome mark indication

2.1.1.7. Dynamics

These articulations involve the loudness a note or musical phrase has to be played with, usually to emphasize certain aspects of the abstract meaning of the composition. They serve as well to identify gradual or sudden changes in volume and they always are subject to the interpretation of the musician as there are not concrete standards established for them. In table 2.1 there is an example of the most used dynamic modifications.

Notation	**Description**	**Marking**
Pianissimo	Play very softly	*pp*
Piano	Play softly	*p*
Mezzo piano	Play moderately softly	*mp*
Mezzo forte	Play moderately loudly	*mf*
Forte	Play loudly	*f*
Fortissimo	Play very loudly	*ff*
Crescendo	Getting louder	<
Diminuendo	Getting softer	>

Table 2.1: Dynamics

There is also the concept of articulation, which also identifies the way the music has to be played. However, one articulation only identifies this for a single note, not for a group of them. Figure 2.21 shows the most common articulations used (from left to right: *staccato, staccatissimo, marcato, accent,* and *tenuto*).

2.1.2. Poetical concepts

In the following, we will explain those concepts related to poetry that are necessary to understand the entire project.

Figure 2.21: Articulation examples

2.1.2.1. Syllable

A syllable is, by definition, a unit of pronunciation that forms a whole word or a part of it. It is generally composed by a vowel sound and either none or several consonant sounds, but this composition varies among languages and accents.

2.1.2.2. Verse

A verse is one of the parts that a poem or song is divided into. The following poem has four verses:

(1) Blind, as was Homer; as Belisarius, blind,

(2) But one weak child to guide his vision dim.

(3) The hand which dealt him bread, in pity kind—

(4) He'll never see; God sees it, though, for him.

2.1.2.3. Rhyme

A rhyme is defined as a word that has the same last sound as another. Poems written in English employ the following types of rhyme:

- *Perfect Rhyme*: A perfect rhyme is a case in which two words rhyme in such a way that their final stressed vowel, and all subsequent sounds, are identical. For instance, sight and light, right and might, and rose and dose.
- *Syllabic Rhyme*: Bottle and fiddle, cleaver and silver, patter and pitter are examples of syllabic rhyme: words having a similar sounding last syllable, but without a stressed vowel.
- *Imperfect Rhyme*: Wing and caring, sit and perfect, and reflect and subject are examples of imperfect rhyme. This is a rhyme between a stressed and an unstressed syllable.

2.1.2.4. Stanza

A stanza is a group of lines of poetry forming a unit. The following poem has two stanzas:

(1) O mistress mine, where are you roaming?

O stay and hear! your true-love's coming

That can sing both high and low;

Trip no further, pretty sweeting,

Journey's end in lovers' meeting-

Every wise man's son doth know.

(2) What is love? 'tis not hereafter;

Present mirth hath present laughter;

What's to come is still unsure:

In delay there lies no plenty,-

Then come kiss me, Sweet and twenty,

Youth's a stuff will not endure.

2.1.2.5. Metre

Metre is the regular arrangement of syllables in poetry according to the number and type of beats in a line. Metrics can also be used to classify poems according to the number of syllables or accentuation.

2.1.2.6. Poem

A poem is defined as a piece of writing in which the words are arranged in separate lines, often short and ending in rhymes, and are chosen for their sound , beauty and images and ideas they suggest. To make them sound better, poems are usually divided into fixed length lines in which each line has the same number of syllables.

2.1.2.7. Types of poems

There are a large number of types of poems depending on their rhyme and structure. Some ways of classifying them depend on:

- **Metre**: They are those of *minor art*, those of *major art*, the *compounds* and the *versicle*. The *minor art* is formed by verses with up to eight syllables; the *major art*, on the other hand, is formed by verses with nine or more syllables. *Versicle* is an irregular verse with no fixed number of syllables and generally so long that it overflows the major art. The verses of more than eleven syllables, called major art *compositions*, have a constant inner caesura or pause fixed towards their half.

- **Number of verses**:

  1 line – Haiku form: Monoku

  2 lines – Couplet

  3 lines – Tercet / Triplet / Haiku

  4 lines – Quatrain

  5 lines – Cinquain / Tanka

  6 lines – Sestet / Sexain/ Stanza

  7 lines – Septet / Rondelet

  8 lines – Octave / Rondeau

  9 lines – Stanza Spenserian

  10 lines – Keatsian Ode

  11 lines – Roundel

  12 lines – Scottish Stanza

  13 lines – Terza

  14 lines – Sonnet / Stanza Onegin / Terza

- **Combining stanzas and/or verses**: *Lira*, the second and fifth verses are hendecasyllables and the first, third and fourth are heptasyllables; *Sonnet*, composed of fourteen verses of major art, hendecasyllables in their classical form. The verses are organized into four stanzas: two quartets (stanzas of four verses) and two tercetes (stanzas of three verses); *Romance*, an indefinite series of verses, in which the even numbered lines have a near rhyme (assonance) and the odd lines are unrhymed lines, in which the even numbered lines have a near rhyme (assonance) and the odd lines are unrhymed lines.

2.2. Lyrics Module

The field of the generation of texts by artificial intelligence has undergone a boom in recent years, and in particular poetry has also been an important topic. It can be considered that its first origins occurred in the 1960s, when some sonnets were created through the recombination of different verses from other poems (Bailey, 1974). Throughout these years different researchers have made important contributions. Some of them we will see in this chapter, although we will try to mention only those that have been interesting or have influenced in some way in our project.

Most of the contributions highlighted below are focused on the generation of poetry and not on song lyrics. However, due to the similarity that may exist between these two kinds of texts, this part of the project has been based on the generation of poetry although it has been given more freedom in certain points to approximate the result to the lyrics of a song.

Since poetry generation is not limited to just text generation but encompasses several parts and processes, we have differentiated between two concepts: *strategy* and *generation technique*.

The strategy talks about the whole process that follows a generator, where among other things the text generation technique is included. We speak of the generation technique as the algorithm, system or model that generates the text.

2.2.1. Strategy of poetry generation

When we talk about strategy in the generation of poetry, we are talking about the different steps that the whole generation process has. This includes the decision to carry out different iterations until returning a result, decide if the model learns from the results it is obtaining or not, etc.

The two methods that influenced our project and that we are going to describe include, on the one hand, a complete process and, on the other hand, a method of adjustment of results and processes of generation of texts used in numerous researches.

In the specification of the objective of the text generators, one of the first decisions we have taken is the definition of the desired result. In other words, *what we want to generate*. Among these possibilities, we find the generation of a single text of great quality, generate several texts and decide which is the best, or generate a first text and improve it little by little until you get something of quality. In the first two options, we find the need to predefine beforehand what we consider a high quality poem. This has been a problem for many researchers, as it is mostly a subjective concept. However, the last option, also called *Case-based reasoning technique* (see 2.2.1.1), allows us to progressively improve the results without the need to achieve the expected result as it learns from past valid poems.

When it comes to adjust the generation of texts, we find the possibility of including some restrictions in the generation process itself, the possibility of making changes to adapt to the restrictions at a later stage, or, as in most cases, a combination of both. Usually, this is stablish by the use of *fitness functions* (see 2.2.1.2).

2.2.1.1. Case-based reasoning technique

Case-based reasoning (CBR) is the process of solving new problems based on the solutions of similar past problems (Aamodt and Plaza, 1994).

These technique exploit path solutions for solving near similar problems in a four-step approach: *retrieve*, *reuse*, *revise* and *retain*. Specifically, in poetry (Gervás, 2001) it is explained in the following way:

> "*Retrieve* vocabulary and line examples that suits fragments of a poem draft; *reuse* the part-of-speech structure of the example lines for producing new lines and combine them with the words in their vocabulary; percent the result in draft to the user, for *revision*; perform a linguistic analysis of the *revised* poems and retain it for further generations."

2.2.1.2. Training through fitness functions

Fitness functions are known to be used in genetic programming and genetic algorithms to guide simulations towards optimal design solutions (Levy, 2001; Manurung, 2003). A fitness function is a particular type of objective function that scores *how close a given design solution is to achieving the set aims*. However, for this project, these fitness functions has been adapted as a way to introduce the desired restrictions on the generation of poetry.

These fitness functions are used to give the text greater poeticity and in general higher linguistic quality. In particular, they allow you to better adapt the rhyme, to look for one or other sounds in the form of vowels or to give more randomness to the text in search of metaphors among others. This is done by prioritizing certain words, with the desired characteristics, over others during the generation of text through Markov Chains (see 2.2.2.2).

2.2.2. Techniques of poetry generation

When we talk about the technique used for generation, we refer to the algorithm, system or model that this generator follows to only create the text. The techniques we are going to cite are: generation based on templates, generation through neural networks, recombinations, Markov Chains and genetic algorithms.

2.2.2.1. Based on templates

This process consists of filling in a template with new words. This template can be a predefined template, or be a template extracted from a text (Colton et al., 2012). This template establishes a syntactic structure that must be filled in with the appropriate word type. The group of words used to fill this template can go from a selection of synonyms words related semantically to the original word that was placed in that position, to the study of the most used words in the body from which the structure has been extracted to guide the generation process. Within this process, we find systems that add additional steps in which, for example, a specific word is substituted to achieve the desired result (Yan et al., 2013). We must also mention that in many cases it is not possible to substitute a single word for another in the text, since the set of words must be taken into account in order for them to make sense. This usually happens, for example, in comparisons, where a single word does not have the same meaning itself, that has when combined correctly with the rest of words forming that comparison (Oliveira, 2012; Oliveira and Alves, 2016).

2.2.2.2. Based on Markov Chains

If a process is divided by states and the probability of the following state X_{n+1} depends only on the previous state X_n and not on all the previous states, we are talking about Markov Property. So to speak, it is a process without memory where past history is irrelevant. Those chains that fulfil this property are called Markov Chains.

From a mathematical point of view, we can say that:

$$X_0 = x_0, X_1 = x_1, ..., X_{n-1} = x_{n-1}, X_n = x_n$$

This technique (Barbieri et al., 2012) allows to, given a corpus, generate in real time verses that satisfy structural constrains.

2.2.2.3. Based on Deep Neural Networks

This process consists of training a neural network to generate texts. The most basic model would be that given a sequence of words, the neural network is able to predict the next one (Potash et al., 2015). The next step would be the generation of new verses or sentences considering the structure and semantics of the poem to be generated (Zhang and Lapata, 2014). We can also highlight the methodology of using two different neural networks to, on the one hand, generate the structure of the lines and, on the other hand, guide the generation of the words (Yan, 2016). Finally, we would also highlight the use of neural networks to measure the fitness of poems generated by an evolutionary approach (Levy, 2001).

2.2.2.4. Based on recombination of different texts

This process consist in the acquisition of full lines of fragments from human-created poems and recombine them in new poems (Malmi et al., 2016; Wong et al., 2008; Charnley et al., 2014). In this process, we find the advantage that the difficulty of finding syntactic or semantic coherence is considerably reduced. However, this can be considered one of the least creative methods since it does not have that point of randomness that allows the machine to innovate in its creations process which can result to literary figures, new structures and rhymes, and verses not seen before.

2.2.2.5. Based on Genetic algorithms

Genetic algorithms are search algorithms based on the theory of the Darwin's evolution. Facing a problem with genetic algorithms involves seek a solution through selection, crossing and mutation mechanisms similar to those employed in nature, so that the most suitable survive (figure 2.22).

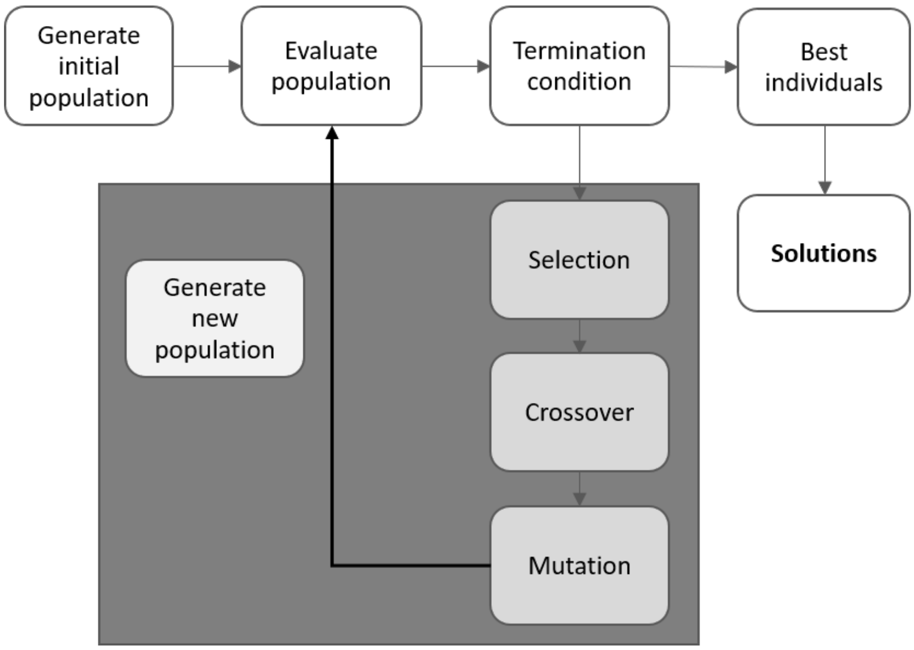

Figure 2.22: General scheme of a Genetic Algorithm

With this approach (Manurung, 2003), texts with syntactic coherence are created correctly with some imposed meter patterns. Nevertheless, its general meaning is not very coherent.

2.3. Evaluation

In this part we will see what a song is, as well as the different approaches that have been taken until these days regarding the automatic composition of them. We will study how songs are typically made and the different restrictions and considerations a songwriter has in mind when composing. We will focus on the various approaches developed to automatically evaluate the computationally composed pieces and parts as well as the different metrics considered for these evaluations.

As stated in the Oxford Dictionary a song is

> "a short poem or other set of words set to music or meant to be sung" [1].

Based on this definition, it is safe to say that a song can be divided into two different parts: lyrics and music, where the lyrics are typically a poem if the are considered just by themselves.

Unfortunately, not a lot of focus has been yet put on the design and development of systems that generate songs automatically in the field of computational creativity. However, many researchers have employed their resources to study the automatic composition of both poetry and music independently, and thus, this will be where our work will be mainly based on.

2.3.1. Evaluation of computer-generated poetry

Poetry generation is one of the most popular fields of study among the computational creativity research community as it not just the generation of text, but the generation of a text that satisfies different aesthetic restrictions. The different techniques used for this generation have already been explained, but now we will focus on the different ways the results of the various applications have been evaluated.

2.3.1.1. Identification of features

As poetry is a very subjective mode of writing, the automatization of the evaluation process of any type of poem (either generated by a computer or by a person) is a very

[1] `https://en.oxforddictionaries.com/definition/song`

difficult task. A poem has to satisfy some features regarding its form and also its content, and this last part is the hardest to study.

Study of form Regarding form, the most important attributes of a poetic text are metre and rhyme as these features are the main reason a given text is recognized as poetic.

- The metre of a poem usually comes from the number of syllables each verse has. There have been different approaches taken when working with syllables, as not all languages follow the same rules. In the case of English, as it is a phonetic language, the division of words into syllables is usually done by means of pronunciation dictionaries as seen in (Manurung, 2003), whereas in latin languages such as Spanish or Portuguese (Gonçalo Oliveira et al., 2007; Navarro-Cáceres et al., 2018) as they are less phonetic, usually defining a set of orthographical rules is enough.

- Rhyme follows a similar approach, as it is based in the repetition of sounds (Oliveira, 2017), and these sounds differ from one language to another. Additionally, the pronunciation of some words may vary depending on the accent they are read with (UK English vs. US English), which poses an extra difficulty to the study of the rhymes in a poem.

Study of content According to (Manurung, 2003), form as explained before is what gives a poem the sense of poeticness, but that alone is not enough to be able to identify a text as a form of poetry. In his thesis he defines two additional features that the poem has to have for it to be identified as that:

- Meaningfulness: it is a property not only applied to poetry, but to any form of text. It states that this text must transmit a message intentionally and it has to be meaningful under some interpretation. Having this property in mind, we are able to determine whether a text can be identified as poetry or not, which is one of the first steps that have to be taken when both generating and evaluating automatically a text that is supposed to be a poem.

- Grammaticality: every poem has to fulfill a set of rules and conventions given by the grammar of the language it is written in. This is necessary for the text to be understandable, even though poetry is known to have some sort of freedom given by the different poetic licences the author (or software) has decided to use. Considering this feature is important to be able to detect overcomplicated sentences that are not understandable for the reader and thus, linking it with the previous point of meaningfulness.

2.3.1.2. Evaluation techniques

Now that we have seen some features to look in a poem to be able to evaluate it, it is time to focus on the different ways we can consider these features and make the study.

Human judges As it was said before, poetry is probably the most subjective way of writing text as its main purpose is to evoke feelings to the reader, and every single person has a different taste. Having this in mind, there have been many approaches to the evaluation of poetry based on the opinions of human judges as it can be seen in (Yan et al., 2013; Yan, 2016; Toivanen et al., 2012; Monteith et al., 2012a). These people were presented a set of computer-generated poems and they had to assign them a score depending on their own personal taste, what provides the system with the point of subjectivity it was lacking due to the difficulty to implement a program that automatically does that.

Turing test Another widely used method to evaluate the results of a poem generator is the Turing test evaluation[1] (Toivanen et al., 2012), which consists on comparing a computer-generated poem to a human-created one. This way, the closer both poems are in terms of perceived quality and likability to the judge, the better the computer-generated poem is, because the human created poem is taken as a good solution. However, there is a downside to the Turing test-like evaluation as explained in (Pease and Colton, 2011), because it just takes into account the result, ignoring completely the creation process, which can lead to the use of very simple procedures to aim to trick the human judge into thinking the poem has been written by a person.

Use of metrics A different approach to take for this process of evaluation is the definition of metrics that measure the most objective parts in the poem (usually related to the form). In the case of (Oliveira et al., 2017), three different dimensions are enough to test the poem:

- Poetic features: form features explained above, such as metre and rhyme.
- Structure variation: to test that the system outputs a different result every time it is given the same input.
- Topicality: the main theme of the poem has to match in some way the seed words provided.

Considering all these features, each sentence is assigned a numerical score, which will show whether it is a good or a bad line, and therefore the system can decide which lines have to be processed again.

[1] `https://en.wikipedia.org/wiki/Turing_test`

Evaluation functions As introduced in Manurung's implementation of an evolutionary solution for the generation of poetry in (Manurung, 2003), the use of evaluation functions that emulate the process human authors use when composing is maybe the most sensible solution to this problem. These functions carry different tests on the generated poems to determine whether their quality is good enough or not, comparing the result to the target metre and semantics.

In line with what Manurung developed, Gervás defined a solution in (Gervás, 2013) and later expanded it in (Gervás, 2013) in which he used the concepts of evaluation functions. In this case, these functions assigned a score to the poem depending on specific parameters (poem length, verse length, rhyme, stress patterns, similarity to the sources, etc.). Having these scores, he developed a module that he called *reviser* whose task is to check the poem generated and determine whether they need to be modified in any way.

2.3.2. Evaluation of computer-generated music

In the case of music, the evaluation process of an automatically generated composition is more delicate, as its study is not as direct as with the poem. The main difference between these two is that words have a meaning by themselves, and it is possible to extract the meaning of a sentence more empirically, whereas with music this process is more subjective.

However, some approaches to the evaluation of music have been already studied and can provide some background for the purpose of our work.

2.3.2.1. Fitness functions

As we have seen, one of the approaches followed for the automatic generation of musical pieces is by means of genetic algorithms. An important feature of these types of algorithms is the *fitness function*, which is an objective function that determines how far an outcome is with respect to a desired solution.

There are different types of fitness functions, as defined in (Phon-Amnuaisuk et al., 2007). These functions can be either dynamic (variable in time) or stationary, and local (evaluations are performed in a local level), or global, and also a combination o both. The type of function is decided based on the programming capabilities of the designer, and also on the good understanding of the domain knowledge (in this case, music composition).

As it is explained in the work of (Alfonseca et al., 2006), the main obstacle to define a fitness function in the case of the composition of music is how to define the features to focus on, and their respective metrics. In this same paper, they explain an already existing function that summarizes all the possible features called the *normalized information distance*. This function is universal as it always measures the same distance between two objects as any other computable metric.

Using this normalized information distance function inside the fitness function defined in (Alfonseca et al., 2006), they develop a system that maximizes the number of features that the generated music individuals share with the guide set, and therefore, the outcome would resemble this set.

Additionally, in (Phon-Amnuaisuk et al., 2007) they define a set of features to consider in the fitness function regarding the identification of the genre of a song, which are mainly the style and metrics of the composition.

2.3.2.2. Human judges

Similarly done as with automatically generated poetry, the most used technique to evaluate the results of music composition is asking for a personal opinion to human judges. In the end, quality of the music is generally decided based on how many people like it, and how much they do.

As it can be seen in works like (Alfonseca et al., 2006; Conklin, 2003; Monteith et al., 2012b), the outcomes of the systems developed are always subject to the evaluation of their developers before they are provided to the potential audience. Once the results are considered good enough, they are shown to people foreign to the development and they are requested to provide their opinion. For example, in (Navarro-Cáceres et al., 2018) the external judges are asked questions about the niceness of the melody, the feelings it provokes on them and how good the rhythm fits the intended genre, among others. After a considerable number of people answered the survey, the average score they provided was studied in order to find the ways the program could be improved.

2.3.2.3. Neural network evaluator

There have also been evaluation approaches taken in the case of music generation based on neural networks. In the work of (Monteith et al., 2012b) they explain how they implemented a hidden layer in their multilayer perceptron system to evaluate the generated selections.

The evaluation component they developed returns a set of statistics obtained after the study of the music, including tonal deviation, number of consecutive identical pitches or rhythmic values, among others. This component also includes an evaluator that compares the given result with the previously generated and evaluated ones, resulting in a system that learns to emulate melodies that have already been accepted by human listeners.

Finally, in their article, Moneith and Martinez state that their version of the evaluator is primitive as finding good features for use in the classifiers is challenging, and that some improvements have to be done in this field in the future.

2.3.2.4. Other significant approaches

In (Herremans and Chew, 2016), a function is defined aiming at the optimization of the results produced by an automatic compositor based on recurrent pattern constraints. This function computes the Euclidean distance between two compositions and then tries to minimize it by improving the solution iteratively.

Although the main purpose of the work in (Gouyon et al., 2004) is not evaluating an automatically generated music, but identifying the genre of a song, the approach followed for the study is interesting for the design of our application. They propose a modular evaluation, first by evaluating attributes of the music individually, and later by studying in subsets, which provides a wider and more complete perspective to the evaluation process.

In (Pearce et al., 2002) the concept of an *acceptability algorithm* is mentioned. The purpose of this algorithm is to study the style of a composition and decide if it can fit inside a predefined corpus of pieces. This is initially just regarding overall style, but it is possible to extract some useful information from how this algorithm deals internally with the music.

2.3.3. Evaluation of computer-generated songs

The fields of automatic generation of poems and music separately have been widely studied as we have seen above, but there have not been many approaches regarding the composition of both at the same time. However, some studies have been made that work with an input music and generate lyrics automatically (Gonçalo Oliveira et al., 2007) and also the integration of this application with another one that generates music automatically (Navarro-Cáceres et al., 2018), with a similar motivation as the one that drives this study.

2.3.3.1. What comes first, music or lyrics?

The first question a person that wants to write a song faces is whether he/she has to first focus on the music and later on the lyrics or the other way around. Nowadays the most important feature many songwriters look for in a song is that it is catchy, that people can identify it quickly. This is achieved from the rhythm, which is mainly given by the music, although lyrics play an important role as well.

In the different studies that have been made in the past (Gonçalo Oliveira et al., 2007; Navarro-Cáceres et al., 2018) music has been taken as the base of the application's flow, and then lyrics are generated on top of it.

2.3.3.2. How to join music and lyrics

As stated in (Gonçalo Oliveira et al., 2007) the first thing that comes ot mind when trying to match some words to a given music is associating each syllable in a word to a single note. The most difficult part is finding a proper way to divide a word into syllables as it is not a trivial matter, and it is a procedure that would have to be studied for every single language individually. In his work, (Gonçalo Oliveira et al., 2007) develops an algorithm to improve what was found in (Velloso, 2006) which achieves this syllable separation in Portuguese words.

When matching melodies and lyrics, different rules can be considered in order to get a better solution. The most important rule to follow is that the strongest beats in the melody have to have assigned the stressed syllables in a word (Hayes and Kaun, 2001), which adds another complication on top of the syllable division process, as it requires a finer study in the language processing technique. Having this in mind, a process was developed in (Gonçalo Oliveira et al., 2007) to determine which words had to be assigned to which notes. This process checked the stress of both the beats and the words, also considering the rests and the pitch repetitions in music.

Additionally, there was an approach to compose music in Spanish (Navarro-Cáceres et al., 2018), which used a system that composed popular music and the already mentioned Tra-la-Lyrics (Gonçalo Oliveira et al., 2007). In this application they implemented the stress pattern study considering the typical structure of already existing popular Spanish songs in form and meaning.

2.3.4. Conclussions

As it has been shown, several approaches have been taken for the composition of both music and lyrics, but not many researchers have tried to combine both of them at the same time. Some applications that generate lyrics for a given melody were developed with a relatively satisfactory result, but the purpose of this project is to take the complexity of this composition a step further and to be able to automatically generate an entire song. However, all these previous work serves as an inspiration for the development of this project as many small details have been taken into consideration in the design and implementation processes.

Chapter 3

Tools

In this chapter the different tools and utilities that have been used for the development of the solution will be enumerated and explained, as well as the reasons why these were chosen before others.

3.1. Programming language and libraries

The project has been developed entirely using the Python[1] programming language, mainly due to the wide variety of open source libraries available, as well as the community that it has behind. It is also a language commonly used when working with large amounts of data, so it was determined that it would be positive for both music and lyrics composers.

The environment used was Anaconda[2] because it comprises several utilities that were valuable for the purpose of the development procedure. For designing and debugging functions independently, the *Jupyter Notebook* tool was used because of its simplicity and the convenience for evaluating results in its interface.

The main libraries used in the solution are explained in the following subsections.

3.1.1. NLTK

NLTK[3] is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning and also wrappers for industrial-strength NLP (Natural Language Processing) libraries.

[1] `https://www.python.org/`
[2] `https://www.anaconda.com/`
[3] `https://www.nltk.org/`

As one of the most complete libraries, it was used not only for the morphological classification of words, but in a first approximation it was used as a syntactic analyser. However, for this last use it was discarded as it did not present coherent and meaningful results.

3.1.2. spaCy

SpaCy[4] is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython. The library offers statistical neural network models for English, German, Spanish, Portuguese and French among others, as well as tokenization for various other languages.

This last feature is the one used in this project to carry out the tokenization process. At first, NLTK was the tool used but spaCy, on the one hand, has much better performance (as shown in figure 3.1) and on the other hand, constructs a syntactic tree for each sentence, which gives us much more information about the text.

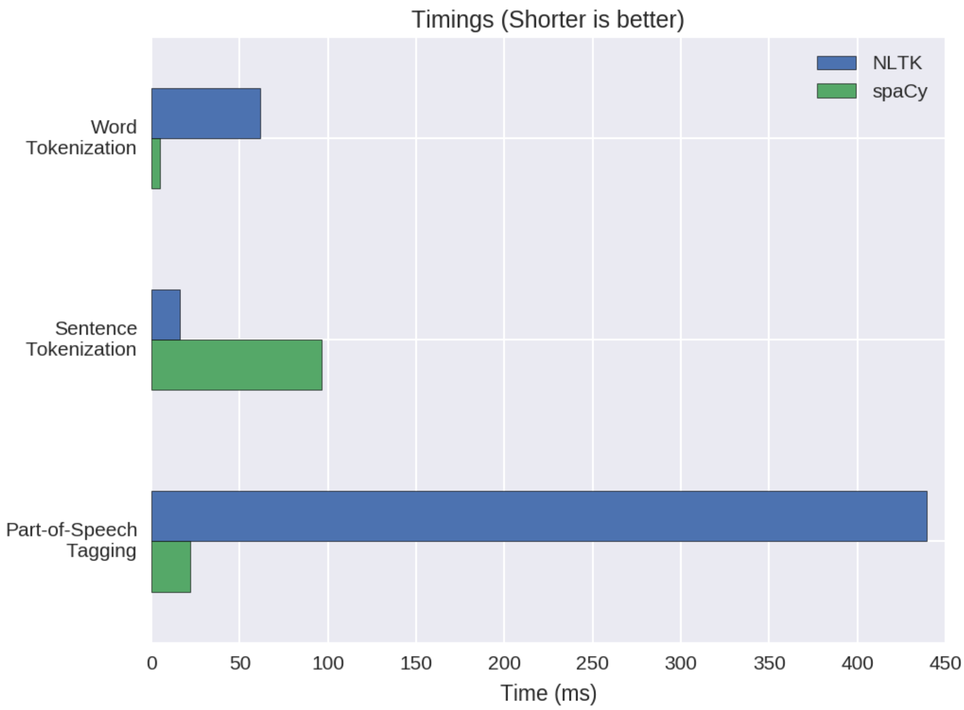

Figure 3.1: SpaCy vs NLTK

[4] https://https://spacy.io/

3.1.3. Thinc

Thinc[5] is the machine learning library powering spaCy (see 3.1.2). It features a battle-tested linear model designed for large sparse learning problems, and a flexible neural network model under development for spaCy.

3.1.4. Syllables

Syllables[6] is a fast, simple syllable estimator for Python. It's intended for use in places where speed matters. This velocity is the reason why we have used this module instead of others during the generation process.

3.1.5. Textblob

TextBlob[7] is a Python library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

In the project, this has been used to perform sentiment analysis due to the simplicity in its use, variety of results according to the configuration you define and the coherence of them.

3.1.6. Pyphen

Pyphen[8] is a Python module to hyphenate words using internal or external dictionaries. It is used for the division of words into syllables during the merging process. Its main advantage is the wide selection of dictionaries that it has built-in, giving the option of modifying the language the application works in very easily.

3.1.7. Markovify

Markovify[9] is a simple, extensible Markov chain generator. Right now, its main use is for building Markov models of large corpora of text and generating random sentences from that. But, in theory, it could be used for other applications.

This Python module is used for generating our text and sentences. Some of the reasons why we used this module were: its simplicity of use, ability to generate sentences and modify this process by applying our own rules, and that requires very few external libraries.

[5] `https://github.com/explosion/thinc`
[6] `https://github.com/prosegrinder/python-syllables`
[7] `https://textblob.readthedocs.io/en/dev/`
[8] `https://pyphen.org`
[9] `https://github.com/jsvine/markovify`

3.1.8. Music21

Music21[10] is a library developed by a research group in the MIT (Massachusetts Institute of Technology) to provide users, mainly related to the investigation field, an easy way to study and work with music. It was created mainly to work with classical music pieces, but it has a large repertoire of functions and utilities that make it possible to work with any kind of music as long as the input format is supported.

For the evaluation process this was the main utility used because of its simplicity and intuitiveness to study the already composed music. It has also been chosen due to the ability it has to assign lyrics to a given piece, as well as the wide variety of input and output formats it accepts.

3.1.9. Numpy

Numpy[11] is a package for scientific computing with Python. Among other functionalities, it contains advanced computing capabilities, tools for integrating code in other languages, sophisticated functions and the possibility to use it as a container of large amounts of generic data.

3.1.10. Built-in modules

Additionally to the external libraries that are developed in Python, there are also many built-in functionalities that helped in the creation of our solution.

- **Statistics**: this module contains various functions to compute mathematical, statistical and numeric operations.
- **RegEx**: this module provides regular expression matching operations to study patterns and strings. It is used to work with the lyrics in the preparation process before joining them with the music.

3.1.11. Musescore

Musescore[12] is a free desktop software that helps open music sheets in different formats. It helps musicians compose and edit their songs digitally, but in our case we used it to open the resulting MIDI files in the form of music sheets during the testing of our solution.

[10] `https://web.mit.edu/music21/`
[11] `https://numpy.org/`
[12] `https://musescore.org`

Chapter 4

Development phase

In this chapter we will explain the functionalities our proposed solution has and how we have implemented it. We will also talk about the different paths we have followed during the development and the reasons that have made us choose the final approach.

4.1. General architecture

Since the beginning of the definition phase of the project, the solution has been thought to be divided in three independent parts. These parts were decided to be the two generator modules (music and lyrics) and an evaluator module. We have decided that the evaluator module would work as the main one, the one that would call the other submodules, because it is where the decision step of finishing the process or trying to improve the solution is made.

Having this division in mind, and also the connections between the different submodules with each other and with the user, the structure of the final solution is as seen in figure 4.1. Since the explanation of the music module's development does not fall within the scope of this project, we will not get into detail about it, which is why this part is shaded in the diagram.

4.2. Lyrics Module

In the following, we present the development of the lyrics generation module. We will start by defining the first approximations and results, starting with the generation cycle followed, the reasons for choosing the Markov Chains for text generation, the problems encountered with the structure and syntactic analysis, and ending with the first result developed.

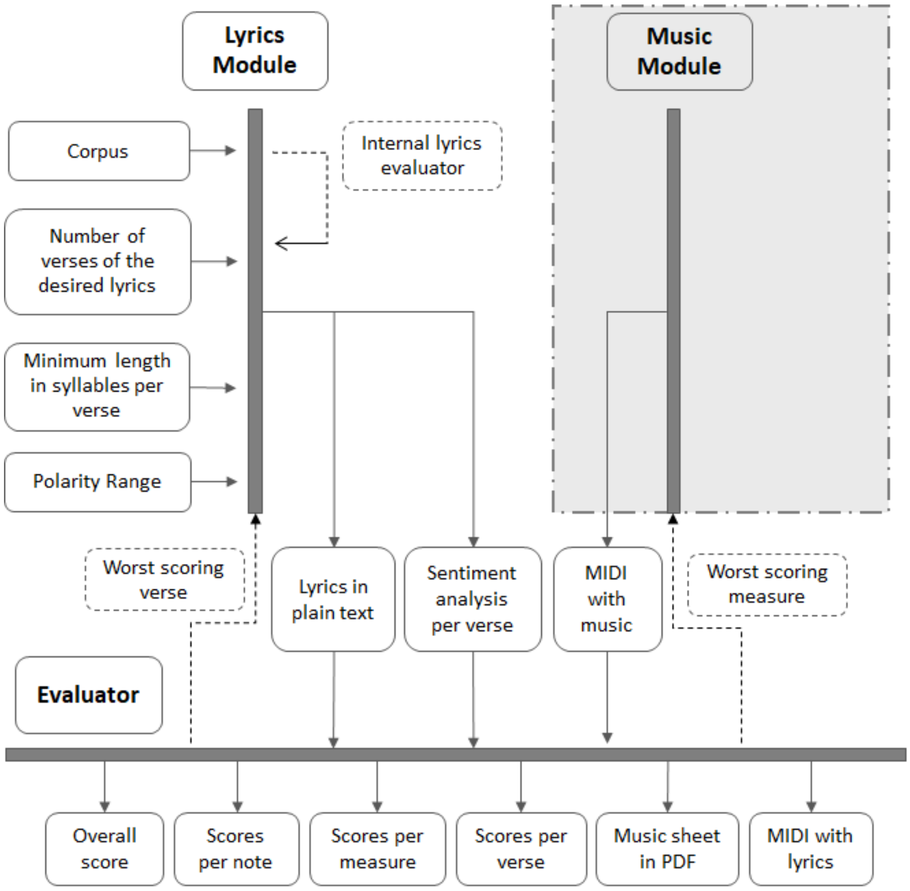

Figure 4.1: General schema of the application

We continue with the specification of the final solution, explaining how we deal with the problems found in the first approach, going into more detail on the feeling analysis and on the external evaluator, and ending with the structuring of the lyrics of the song.

4.2.1. Approach

Our first development is designed in different modules as shown in figure 4.2. Each module has its own utility as explained in this section.

We firstly started by defining our text generation strategy. In the *Case-based reasoning technique* (see section 2.2.1.1), we highlight the fact of reusing parts of other poems, as well as the presentation of a first result to the user for review. In our case, the user only presents the final result without the reviewing step. Therefore, our approach (shown in figure 4.3) could be better defined as: **generate, train, evaluate, correct and save**.

We **generate** a draft text by **training** the algorithm according to a corpus, we **evalu-**

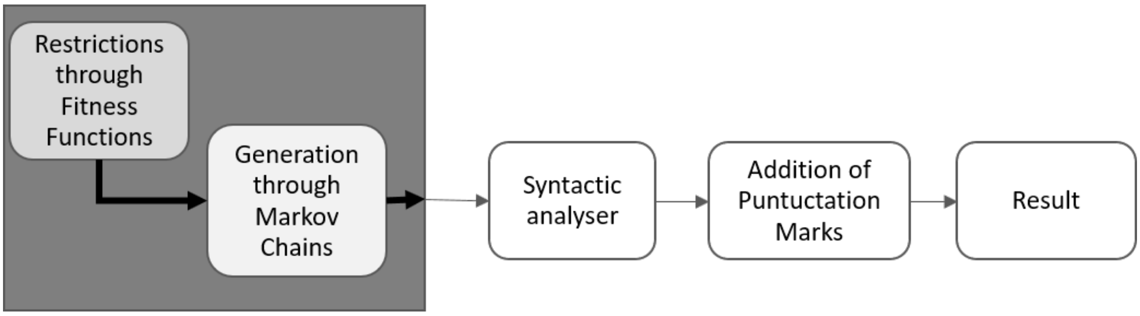

Figure 4.2: Schema of first approach structure

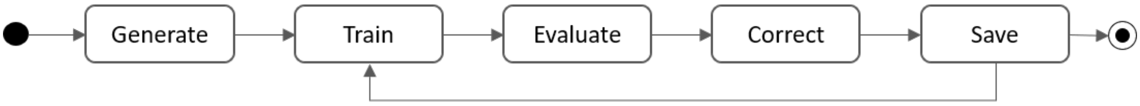

Figure 4.3: Text generation process

ate the result and its quality, we **correct** what we need, and we **save** it for the algorithm to learn and re-train.

During the development of the project, we will understand the **generation** phase as the generation of the text through an algorithm without any limitation; the **training** will take place at the point where it begins to restructure or modify this first training so that, for example, it has a more poetic structure; the **evaluation** will contemplate the external evaluation that will give us those parts to **correct**; and finally we will show the final result to the user and we will **save** it to retrain this algorithm.

4.2.1.1. Technique of text generation

With the techniques described in the previous chapter (see section 2.2.2), we made an assessment of what we consider the most appropriate for our project.

The main considerations we took into account when choosing this technique were:

- Time it takes to generate a poem.
- Capacity to modify the generation process, and not only to modify the final result.
- Presentation of different results in terms of syntax and semantics in each generation.

The technique *based on templates* (explained in section 2.2.2.1) had many advantages when it came to facilitating syntactic coherence in the poem. However, as with the *recombination technique* (see section 2.2.2.4) of different poems, it left very little room for creativity and the final texts generated were very similar to each other in these terms.

The use of *Deep Neural Networks* (see section 2.2.2.3) was discarded due, principally, to the difficulty of adjusting the results and the generation process and also for the time it takes to generate an output.

With *Genetic Algorithms* (see section 2.2.2.5) we achieve a syntactically coherent text but it was too far in terms of global meaning of what we expected. In fact, we discovered in this try that it was much easier to correct the syntactic structure of a text rather than its meaning.

Finally, we decided to use *Markov Chains* (see section 2.2.2.2) for the generation of texts since the generation time is short enough for our purpose, it is easy to adapt its generation to our needs and the outcomes are different in each generation.

A comparison between these considerations and the techniques studied can be seen in table 4.4:

Conditionalities/Needs	Based on templates	Recombination technique	Deep Neuronal Networks	Genetic Algorithms	Markov Chains
Time it takes for generation	Yes	Yes	No	Barely	Yes
Capacity to modify the generation process	No	Barely	Barely	Barely	Yes
Presentation of different results in each generation	No	No	Yes	Barely	Yes

Figure 4.4: Comparison of different generation techniques

4.2.1.2. Markov chains

In the case of the use of the Markov property (see 2.2.2.2) for the generation of texts, the analysis of a corpus is usually carried out by the study of the probability that one word follows another, taking into account only the word immediately preceding it. This property has numerous advantages for our project among which we find:

- It is the most flexible when it comes to introducing new changes and modifying the probability that one word has to be followed by another.
- Generate texts with great similarity to the originals. For some cases of other generators, this can be a disadvantage since you may want to look for an innovation or a novelty when generating texts. However, since our generator is developed to remind to other people's songs, this similarity is an advantage. The most used words or expressions will be more likely to be reused than the others, which will allow the user to get that feeling that the song "sounds like something". An example of this repetitive use by certain artists of some words is Lorca. This poet uses words such as *moon*, *gypsy* and *blood* to a large extent in his poems.

4.2.1.3. Restrictions in the text generation

Once the technique for text generation was decided, it was necessary to define the restrictions we will apply to this process. Some of these points we delivered were:

- Text generator: infinite text generation vs. phrase generations.
- Use of fitness functions.
- Addition of punctuation marks.
- How to analyse the syntactic structure.

Text generator and fitness functions

One of the points was to decide whether the generator will generate a text *indefinitely* based on a corpus, which would then be modified to obtain higher quality and more literary coherence, or it will generate *phrases* and verses in a controlled manner, which later would be modified to obtain that quality and coherence.

In this first approach, we generate the text indefinitely through Markov chains. In this process, we made use of fitness functions with which, for example, we could control the number of rhymes that would appear in the Lyrics or if the text was generated with more or less randomness. These functions allowed us to have more control over the generated text, to be able to give it more poeticity, and in general, to train the generation algorithm in the way that we wanted.

Some of the fitness functions defined can be seen in 4.1.

Syntactic structure

For the analysis of the syntactic structure of the generated texts, we started by using a similar approach to the *based on templates* technique. We considered the option of being able to establish predefined structures that were later being filled with the words with the corresponding syntax. However, the approach was discarded because the results obtained were always very similar in every iteration and gave little place to creativity on the part of the program.

Another of the approaches was to analyse the syntactic structure at the same time as the text was created, so that when we found a word that was out of place in this context, it was modified or moved. However, this approach is very costly both in terms of generation time and complexity.

```
# For less complex words, that is, shorter word lenght, we use:

    def aw_favor_simplicity(a,b):
        return len(set([c for c in a + b])) / len (a + b)


# For having more rhymes in our verses:

    def aw_favor_rhymes(a, b):
        a, b = sorted ([a, b], key = min)
        return sum([1 ip p[0] == p[1] else 0 for p in zip(b[len(b) - len(
            a):],)]) / len (a)


# To get more musicality or the alliteration of one or more vowels in
    particular:

    def  aw_favor_vowels(a, b):
        return sum([1 if c in 'aeiou' else 0 for c in a + b]) / len(a + b
            )
```

Listing 4.1: Code fragment of some Fitness Functions

```
proper_nouns = ('NNP', 'NNPS')
nouns = ('NN', 'NNS')
pronouns = ('PRP', 'PRP$')
verbs = [('VB'), ('VBD', 'VBG'), ('VBD'), ('VBN'), ('VBP'), ('VBZ')]

subjects = [('DT', 'NN'), ('NN', 'NNS'), ('PRP', 'PRP'), ('DT', 'NNS'),
    ('PRP', 'JJ'), ('PRP$', 'NN'), ('DT', 'JJ', 'NN')]
```

Listing 4.2: Examples of predefined subjects and verbs

A third approach was to use the libraries provided by the NLTK library (explained in section 3.1.1) specifically *Treebank Tokens* and *Brown* to syntactically divide the different sentences. However, we have found that the score obtained with automatic algorithm generation itself is not done in a right way, so when dividing a song in its various sentences, we find errors or simply the inability of the functions to do so.

Taking into consideration all these first approximations, we decided to look for a different way to perform this analysis. We decided to make a module that analysed the syntactic structure *a posteriori* and that added punctuation marks or modified its structure, according to certain criteria that we would apply. Some of the predefined structures can be found in figure 4.2.

At this point, we consider how to perform the parsing. When analysing a text morphologically by words, it is quite simple since there are many dictionaries that give you the word morphological class and even analyse it according to the context, if that word is used as for example, a name or an adjective. An example of this analysis can be seen in figure 4.3.

```
Hold me tight so I'll be your lover baby, now

[('Hold', 'NN'), ('me', 'PRP'), ('tight', 'VBD'), ('so', 'RB'), ('I'
    , 'JJ'), ("'ll", 'MD'), ('be', 'VB'), ('your', 'PRP$'), ('lover
    ', 'JJ'), ('baby', 'NN'), (',', ','), ('now', 'RB')]
```

Listing 4.3: Morphological analysis

Therefore, morphological analysis has not been a barrier. However, when we speak of noun phrases or sets of words, the complexity increased considerably. We decided to create our own methodology since we did not get the expected results with the libraries we tested (NLTK). This was based on the creation of three modules that look for different parts of the sentences: *Subject*, *Verb* and *Predicate*.

- **Subject analyser**: The subject analyser looked for the possible subjects of the sentence. If we were working in Spanish, this would be much simpler since we could find out what the subject is by looking for the genre and plurality of noun phrases. However, when doing it in English, this task becomes a little more complex and not as accurate as it should be since there is no single conjugation for each subject as in Spanish. Words belonging to the subject were identified with a '1'.
- **Verb analyser**: On the other hand, we have the verb analyser, which finds the sentences' verbs, both simple verbs and compound verbs. Words belonging to the verb were identified with a '0'.
- **Predicate analyser**: The words that were not from the subject nor the verb, were the predicate. Words belonging to the predicate were identified with a '-1'.

The following example in figure 4.5 shows an approximation of the desired result achieved with our syntactic module. And the code snippet 4.4 shows a real example. However, it is possible to see that this function still has some errors.

4.2.1.4. Punctuation marks

This process completely depended on the syntactic module as it needs the syntactic structure analysis to identify the different parts of a sentences and its endings.

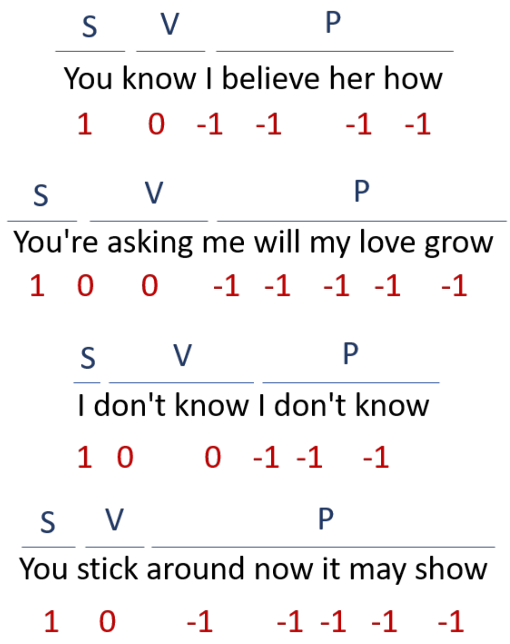

Figure 4.5: Syntax analysis example

If the generator is able to correctly identify the predicates, and taking into account the structure that certain songs of certain groups usually follow, we could venture and establish the beginning and the end of the sentences. This, in turn, will allow us to enter the necessary punctuation marks. In addition, we would identify conjunctions that serve as a union between different sentences, to take them into account when identifying these sentences and, thus, be able to apply punctuation marks in an appropriate way. Also with the identification of verbs, we could identify the number of sentences, whether they are subordinate or not, that are in the text and this also allows us to specify better the number of sentences and therefore of punctuation marks.

4.2.1.5. First result

We finally obtained a generator of songs that had no end, to which the punctuation symbol was removed to be able to analyse syntactically through the NLTK library (see section 3.1.1), and they were analysed by our parser.

At this point, we understood that, even if the parser worked correctly, the usability that it might have was not the most appropriate for our project. In addition, due to the inaccuracy that this analyser had, most of the results obtained were not correct.

At this moment there is a turning point, in which we had to look for another approach to be able to produce quality lyrics. It is decided to look for an approximation that gives us

```
Hold me tight so I'll be your lover baby, now

[('Mery', 'NN'), ('was', 'PRP'), ('sitting', 'VBD'), ('on', 'RB'), (
    'the', 'JJ'), ("chair", 'MD'), ('her', 'VB'), ('sister', 'PRP$')
    , ('was', 'JJ'), ('swimming', 'NN'), ('alone', ',')]

Mery - 1 (Subj)
was - 0 (Verb)
sitting - 0 (Verb)
on - -1 (Pred)
the - 1 (Subj)
chair - 1 (Subj)
```

Listing 4.4: Syntactic analysis module

the punctuation marks in the own generation of songs, that preserve a theme throughout the text in terms of semantics and feeling, and one on which we had a greater control capabilities to limit this generation of verses.

4.2.2. Final result

A resume of the path that has been finally followed is shown in figure 4.6. In addition, an outline of the different modules designed in this final result is shown in the figure 4.7 and is explained in the following subsections.

4.2.2.1. Text generator, syntactic structure and punctuation marks

The path that has finally been followed in this project starts with the use of *Markov Chains* for the generation of text (see section 3.1.7). However, this has been raised in another way. As the syntactic analysis was a great barrier in the previous approach, we have simplified this module of analysis in a module that verifies the coherence between subjects and predicates by sentences.

Previously we had a generator of words, that had no limits on the part of the algorithm when generating sentences. This made the text endless and barely paused, which resulted in a single phrase that could occupy 30 verses. To avoid this, we discovered that in shorter generations, the sentences that the algorithm gave us had more syntactic coherence. For this reason, we decided to establish a generator that would give us sentences with an ending, and not simply one word after another. Thanks to this approach, we can set aside the punctuation marks since, in general, the algorithm itself includes them in the correct way.

On the other hand, we had simplified the analyser of subjects, verbs and predicates.

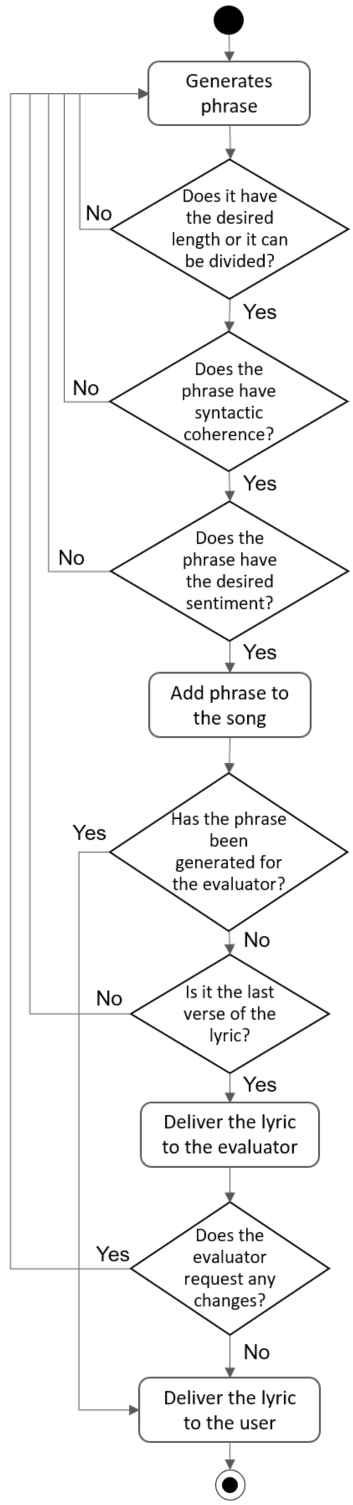

Figure 4.6: Lyrics generator scheme

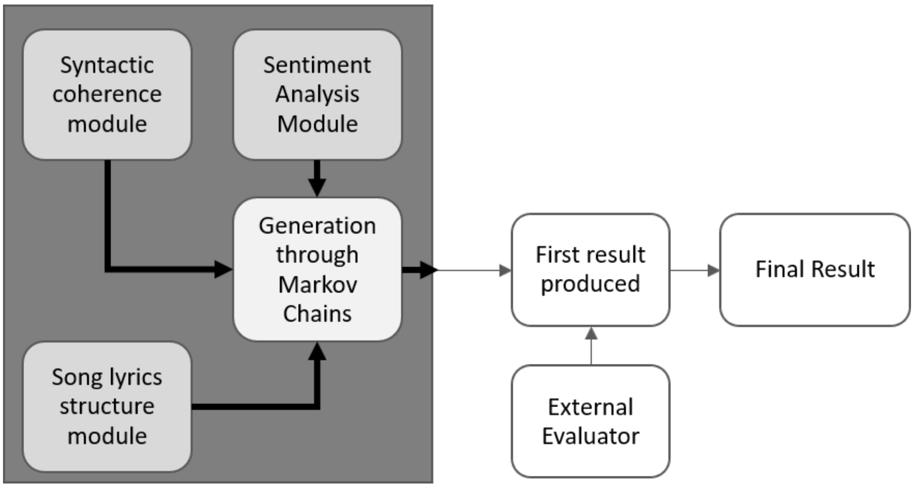

Figure 4.7: Schema of the final result structure

```
[TRUE] - Although I(1) laugh(1) and I(2) don't want(2) you, But I(3)
    need(3) you.
[FALSE] - So please listen(1) to me, if she(2) got(2) no time for you
    right now, don't bother(3) me.
```

Listing 4.5: Example of syntactic coherence module

Since we did not obtain complete reliability in this module, but we did have higher reliability in the morphological analyser, we looked for the way to carry out this syntactic analysis supporting us as much as possible in the morphology of the words. The fact of obtaining shorter sentences has also allowed us to reduce complexity and the need to evaluate the generated text with great precision. That is why this parser, assisted by spaCy (see 3.1.2) seeks to ensure that the subject and predicate have coherence with each other, that is, that there is a subject for each predicate or verb and that it has coherence with it. For example, if a subject is plural, the conjugation of the verb must also be consistent.

This whole process can be followed in figure 4.8.

4.2.2.2. Sentiment analysis and external evaluator

It should be noted that when we speak of sentiment analysis, we speak on the one hand of polarity analysis and on the other hand of subjectivity. The analysis of the feeling is expressed in the following way: the module gives a number in range [-1..1] that indicates the general feeling of the songs in terms of polarity and subjectivity.

- **Polarity**: It lies in the range of [-1,1] where 1 means positive statement and -1

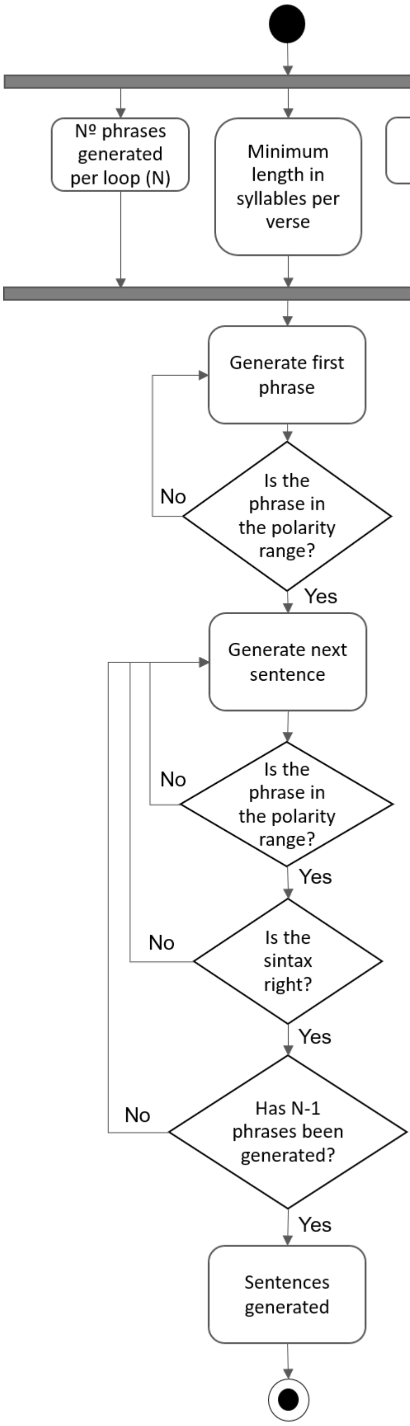

Figure 4.8: Sentences generator process

means a negative statement. It is important to be careful with this field, since if we approach this value too much to -1 or 1, the module chooses verses very similar among

```
>> Analytics Vidha is a great place to learn data science.
Sentiment(polarity=0.8, subjectivity=0.75)
```

Listing 4.6: Textblob sentiment analysis example

themselves, as there exists a smaller number of them with this value of polarity in the battery of phrases.

- **Subjectivity**: It lies in the range of [0,1] where 0 means a very objective text and 1 a very subjective one.

First, this evaluation is done during the algorithm's text generation process. We have added an evaluation for each generated sentence of polarity and subjectivity using Textblob (see section 3.1.5) (example shown in code snippet 4.6). If the phrase has a deviation lower than the one accepted to be introduced in the text, it is included and, if it is not, it is discarded. A schema of this process is shown in figure 4.9

On the other hand, we have the sentiment evaluation of the external evaluator. This evaluator gives us a verse number to modify, and we regenerate that part of the text. This process can be seen in figure 4.10.

We have set the polarity and subjectivity between [[-0.7, -0.25] and [0.25, 0.7]]. The results obtained with a greater or smaller polarity and subjectivity gave us not valid results.

4.2.2.3. Structure of the song

For the structure of the song, we must bear in mind that thanks to this generator by phrases, establishing a structure is much simpler. Through this process (shown in figure 4.11), it is easy to establish the refrains and verses or modify the number of verses by stanza.

In the beginning, quatrain songs structures will be presumed in which we will find octosyllabic verses. The counting of syllables is done with *Syllables* (see 3.1.4). As we are creating phrases, and a phrase can occupy more than one verse, the length of the stanza can vary. We can also tell which stanza to repeat to create the chorus. Our predefined structure is: stanza, stanza, chorus, stanza, chorus.

4.3. Evaluator

As we have seen previously during the research process of the work that has already been done in this field (see section 2.3), the main challenge we are facing here is determining a rather objective way to decide if a given poem would fit in as the lyrics of a music

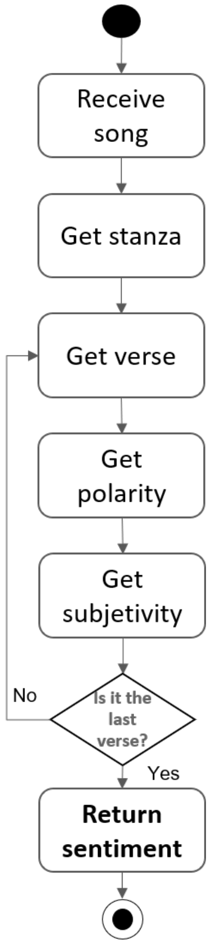

Figure 4.9: Sentiment analysis process schema

composition. This is a very subjective process, usually defined by the personal preferences and choices of the songwriters and composers, who commonly use them to show off their composing and/or performing strengths on the way they choose to match melody and lyrics.

For the process of evaluation the development was decided to be done modularly. Thus, the design of the module was made having in mind the extraction of information from both lyrics and music separately, with the consequent merging process of these two to produce a complete song.

4.3.1. Architecture

The evaluator module was designed to have one main component and two subcomponents, one to process each of the input data received by the two composers. It was developed this way in order to improve modularity by making a clear separation of the two

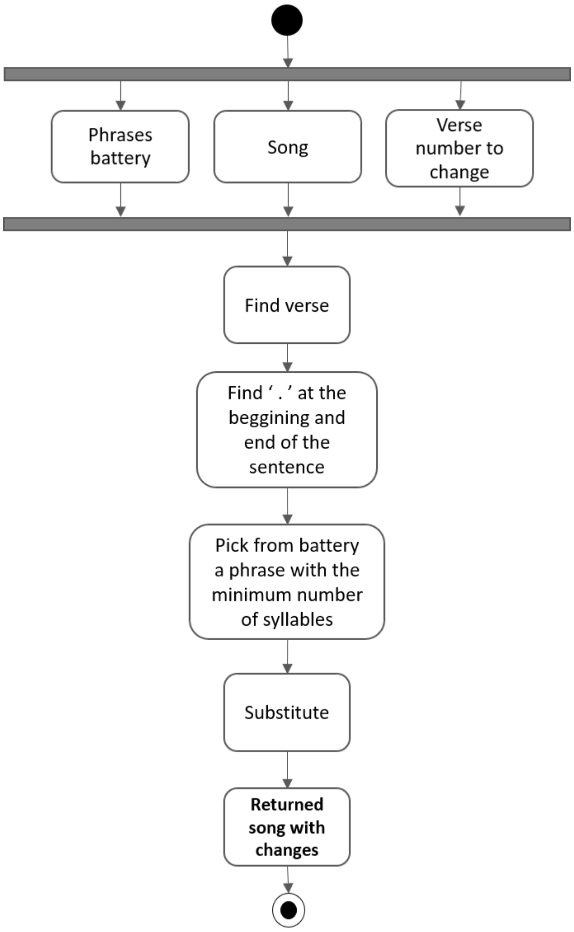

Figure 4.10: External evaluator process

inputs, easing the validation and verification processes.

The main module is called the *Merge module*. It works as an intermediate communication point between the two other submodules and the generators. Therefore, it initially receives all the inputs from both automatic generators and deals with them, first carrying some preparation steps needed in order for the submodules to be able to work with the data, and later sending them to the submodules for them to carry their computations. In the end, the main module will process the outputs resulting from the merging operations and deal conveniently with them (discarding data, sending feedback, returning outputs to the user, etcetera). This structure can be seen in figure 4.12.

The other two submodules are the aforementioned *Information extractors*, one for the music and another for the lyrics, where each of them deals with their corresponding inputs. During their processes, they extract relevant information from their corresponding input and work with it conveniently, extracting important data points, computing relevant values

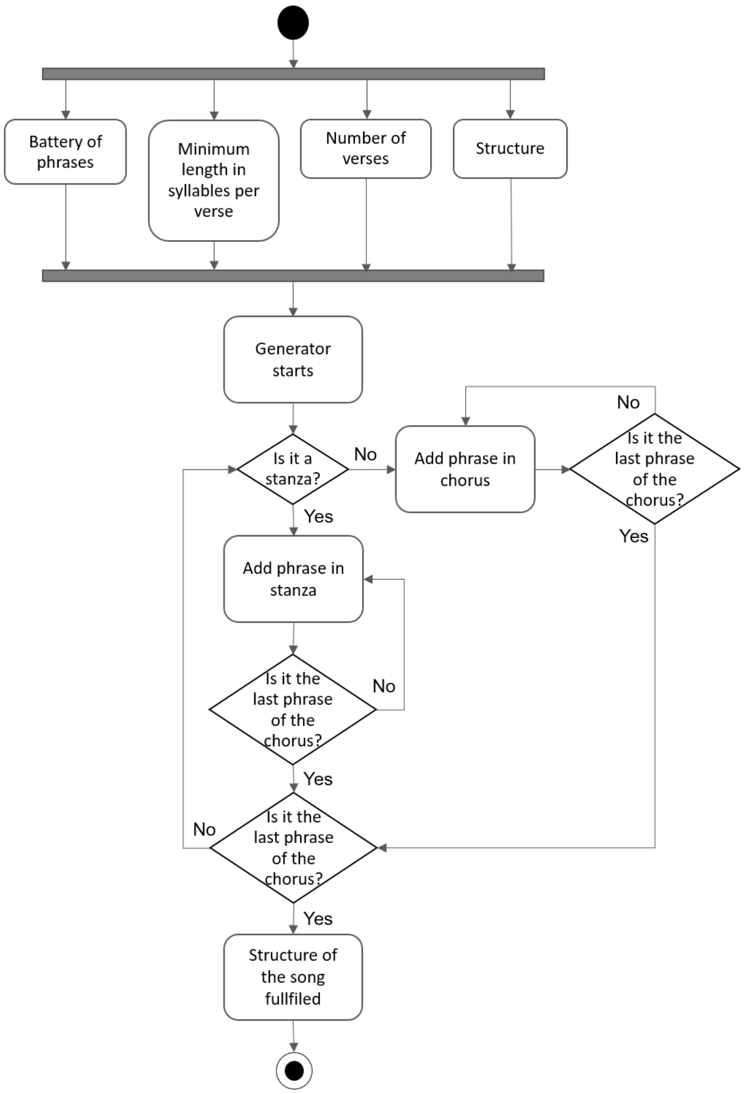

Figure 4.11: Adding verses in structure process

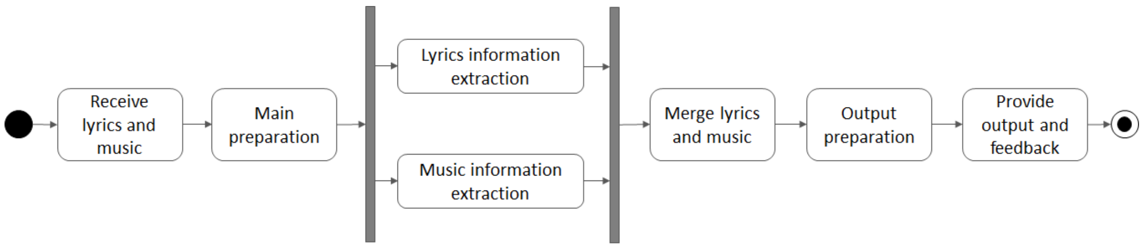

Figure 4.12: General flow of the evaluator module

for the merging process and arranging the variables agreeably for the main module to handle them.

4.3.2. Lyrics information extraction

The first input to be processed is the poem that will serve as the lyrics, and for this it is important to have in mind the different theoretical concepts mentioned in section 2.1.2. To extract all the relevant information from the poem it is necessary to initially prepare it because not all the important data points can be obtained or studied from raw text. Thus, the process was designed to first treat the input in order to be able to work with it throughout all the steps (as seen in figure 4.13), and after this treatment, defining a data structure in which all the information extracted will be stored for further use. Having this structure defined, it is possible to extract all the convenient information and fill in this structure consequently.

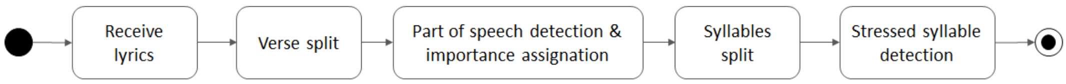

Figure 4.13: Flow of the lyrics information extractor

Having the entire poem by itself, the first step is to divide it into syllables. For this, a previous process of preparation is needed, which consists in cleaning the poem from empty line breaks and splitting it into lines. Each of these lines are going to be the different verses that compose the poem. It will also be useful for the future to keep in the data structure the information of which verse number out of the entire poem each line has, as it will be important during the later merging process with the music.

Once the poem is split into verses, the next processing step needed is dividing every single word in each verse into syllables. This is an essential step because, when the lyrics are being assembled along the music, each syllable is typically sung at the same time a single note is played. For this step it is necessary to determine beforehand the language the lyrics are written in, but as the process is generalized, it would work with any language defined in the Pyphen library (see section 3.1.6). In this case, as the project has been developed in English, this language's dictionary is used.

Up to this point, all the steps performed with the lyrics have been by means of auxiliary libraries that provided the exact functionalities that were necessary. However, there is still more information and data points to be extracted from the poem which will be relevant at the point of merging it with the music. Therefore, a set of different algorithms have been defined, in order to get each of these relevant data points.

4.3.2.1. Part of speech

The part of speech indicates the way a word is used in a sentence. There are several categories of word classes (another way to name these grammatical identifiers), and in every one of them the words share different grammatical, lexical, syntactical and, in some cases, morphological properties. In the English language there are ten main classes, and each one of them can be divided into several subgroups that identify every word more specifically. These main classes are: noun, verb, adjective, pronoun, adverb, conjunction, preposition, interjection, determiner, numerical and article.

After determining the various types of possible word classes, a ranking was designed sorting the different parts of speech from the most to the least important in a sentence. This was made because it is considered valuable to know the relationship between the importance of each word and the note associated to it, so this data point computed would be used further during the merge process. There is not an official list made for this purpose, but it is possible to find many different ways thought on how to sort these word classes, although many of them usually coincide in the most and the least important types of parts of speech in the scale. To settle this ranking, we considered the already created list, but gave a personal approach to it by studying the different characteristics of each one of the categories: the morphological meaning the word has by itself, the syntactical importance of the word in a sentence (how dispensable the word is for the sentence to have sense) and its overall grammatical importance. The flow of this process can be seen in figure 4.14.

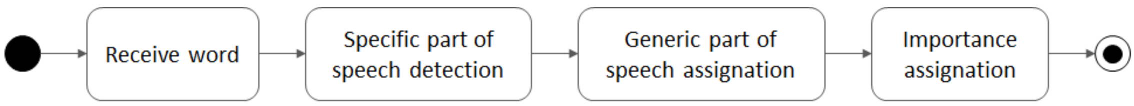

Figure 4.14: Part of speech detection and importance assignation

Therefore, the resulting list is as follows (from most important to least important): *verb > noun > adjective > adverb > preposition > pronoun > conjunction > determiner > interjection > the rest of possible classes*

This	application	generates	beautiful	songs	automatically
det	noun	verb	adjective	noun	adverb

In the example above we can see that when removing certain words from the sentence, the meaningfulness of it decreases. If we dispense "this" the sentence will keep almost the complete meaning, contrary to what would happen when removing "generates" or "songs".

The library used to determine the part of speech of each word is NLTK (see section 3.1.1). It identifies the word in a subclass of one of the main classes, so it was necessary to process the output of this built-in function in order to generalize it to one of the main classes. Additionally, there are some words (not just in English) that can act as different parts of speech, either because they are polysemic (they have more than one meaning) or depending on the grammatical and syntactical structure of the sentence, but for this first approach we trust the NLTK built-in function.

4.3.2.2. Stressed syllables

Considering the definition of a syllable from section 2.1.2.1, we will study the poem and divide it into syllables. However, the division of a word into syllables varies completely depending on the language it is written in, as each one of them has a different set of rules to define this division (in the Spanish language, for example, a syllable can have more than one vowel sound). Working in English we face another challenge as there are different accents where the pronunciation varies for the same word. For example, the word "mobile" is pronounced differently in American English and in British English (RP – Received Pronunciation, the standardized pronunciation in the United Kingdom), as in America it has two syllables (mo-bile /[ˈmoʊ-bəl]/) and in the UK it has three (mo-bi-le /[ˈməʊ-ba-ɪl]/). This acts as a useful example for the definition of syllable, as it does not depend on the way the word is written, but on how it is pronounced.

Knowing what a syllable is and what it looks like in English, we have to go one step further. For lyrical purposes it is almost as important to know how to divide a word into syllables as to know which of the syllables in the word is the stressed one when speaking. It is essential to know this because when associating the lyrics to the music, the stressed syllable of each word should be associated to a note that also has a higher importance than its surrounding ones.

The identification of the stressed syllable in a word poses another challenge as there are not official rules defined to detect this. Additionally, it varies significantly depending on the accent the speaker (or the reader, in our case) is pronouncing. We can find an example for this in the word "advertisement" as, if we divide it into syllables (ad-ver-tise-ment) in British English the stressed syllable is the second one and in American English it is the first one. In this first approach, it was decided that the accent in mind when writing the lyrics was the British one.

To determine which is the stressed syllable in a word we followed the process shown in figure 4.15, in which a set of rules were defined based on several observational examples:

- If the word is monosyllabic (i.e. it is composed by just one syllable) then that would be the stressed syllable. Examples: *go, get, bed.*
- Numbers and words ending in *-ty* usually have the stress in the first syllable. Exam-

ples: *fifty, seventy.*

- Numbers in which the last syllable is *-teen* typically have the stress in the second syllable. Examples: *sixteen, eighteen.*
- Every word that ends in either *-tion*, *-sion* or *-ic* is usually stressed in the second-to-last syllable. Examples: *contextualization, commission, fantastic.*
- If a word is ended by *-cy*, *-phy*, *-al*, *-ty* or *-gy* the stress of the word is in the third-to-last syllable. Examples: *democracy, philosophy, cynical, commodity, dermatology.*
- If it is a bisyllabic word and it is either a noun or an adjective, the stress is usually on the first syllable. Examples: *pencil, handsome.*
- If the word is a verb or a preposition and it is composed by two syllables, the stress will probably be on the last syllable. Examples: *compare, between.*
- If the word is formed by six syllables and its last one is *-tion*, then the stress syllable will probably be the second-to-last one. Example: *initialization.*
- For the words that are not in any of these previous groups, the standard defined is that the stressed syllable is the longest one in number of letters, as we cannot identify the specific phonetic syllables.

Having extracted this information and computed the data points from the received poem, it is possible now to determine if it can fit more or less satisfactorily as the lyrics of a song. As a reminder, we have the **poem divided into verses** and each verse contains all the words in it **divided into syllables**. For each word we know its **part-of-speech** and the **importance** associated to it in the scale previously explained. As an important addition we are able to identify whether **each syllable is stressed or not**, and we also know in which **number of verse** it is in. All this information will be stored and passed to the main module so that it can be used when merging the lyrics with the music and determining the quality of the result. An example of this data structure can be seen in code snippet 4.7, outcome obtained after processing The Police's “Every breath you take” first verse.

4.3.3. Music information extraction

Following a similar approach to the one we have explained before with the lyrics, we have to focus then on the music received. The purpose of this module is to try to identify

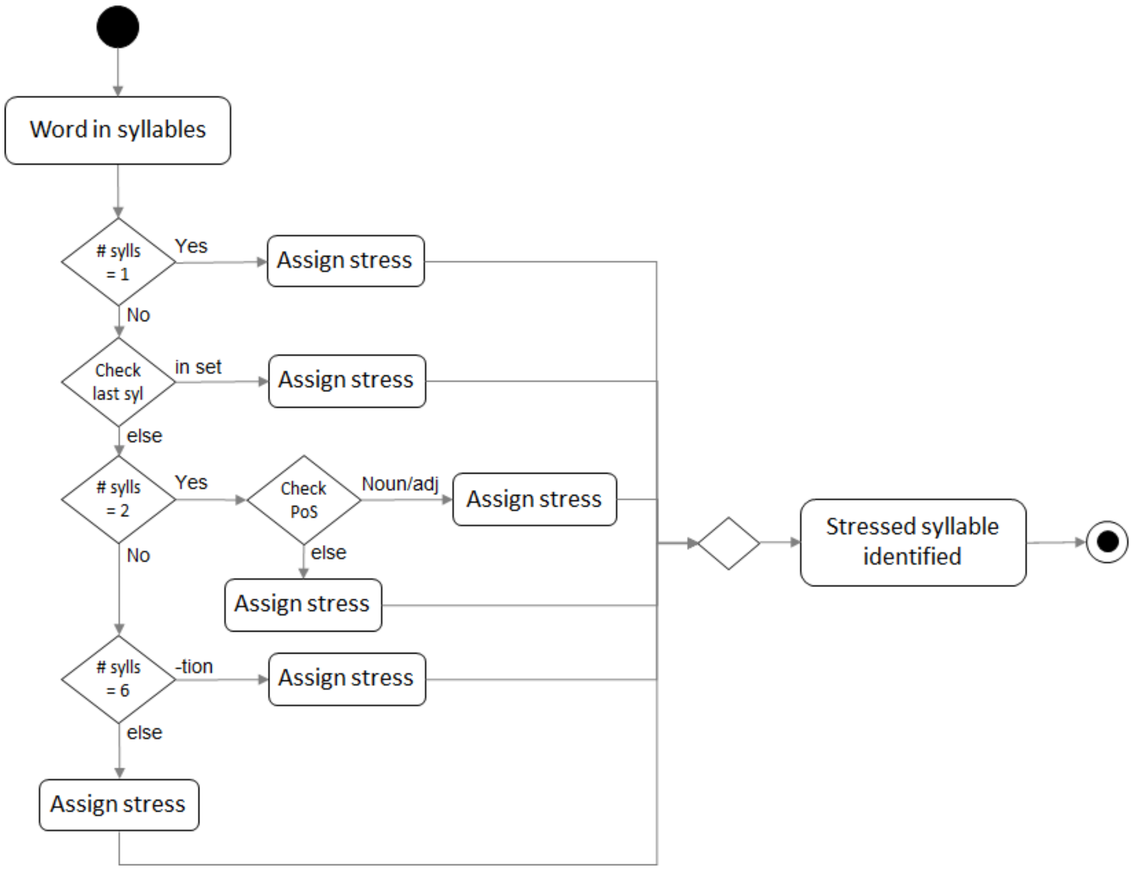

Figure 4.15: Stressed syllable detection process

different characteristics of the composition that can help us determine if the merge of the poem with it is satisfactory enough.

In the initial phases of the development, the standard defined for the music file format was the ABC notation[1] as it was thought to be more convenient for both the composer and the evaluator modules to work with. The use of this notation was later discarded due to the lack of some useful information that would be present by means of other music formats, as well as the capabilities of the utilities found to work with them. In the end, it was decided that both the input and the output of the evaluator module was going to be a file in MIDI format[2] as we considered that it contained the necessary information to be able to evaluate the merging of the lyrics with the music itself, and also generate more manageable outputs.

The tool chosen to work with the music for its evaluation was Music21 (see section 3.1.8) as its functionalities matched conveniently the ones we were focusing on from the

[1] `http://abcnotation.com/`
[2] `https://en.wikipedia.org/wiki/MIDI`

```
[[{'Ev': {   'stress': 1,
             'part_of_speech': 'determiner',
             'importance': 8,
             'verse': 0}},
{'ery': {    'stress': 0,
             'part_of_speech': 'determiner',
             'importance': 8,
             'verse': 0}},
{'breath': {'stress': 1,
             'part_of_speech': 'noun',
             'importance': 2,
             'verse': 0}},
{'you': {    'stress': 1,
             'part_of_speech': 'pronoun',
             'importance': 6,
             'verse': 0}},
{'take': {   'stress': 1,
             'part_of_speech': 'verb',
             'importance': 1,
             'verse': 0}}]]
```

Listing 4.7: Data structure after processing a poem

beginning, and also included some that helped defining additional computations that were not initially considered.

The files in MIDI format have several voice channels, each one associated to a different instrument. This is useful if you just want to focus on a concrete instrument as it is possible to isolate its channel in many media players. Another advantage provided by MIDI files can be opened with different XML readers like Musescore (see section 3.1.11) that are able to show the music sheet as well as play the music, something that was very useful at the beginning for being able to interpret the music as a beginner. The input music for the evaluator module was decided to be composed of just two instruments (a piano and a bass rhythm drum) and hence two voice channels. This way it would be possible to isolate the melody's channel for the sake of simplicity and convenience for this first approach.

From the development point of view, it is not possible to work directly with a file in MIDI format, so a preparation step is needed to parse the input into a format we are capable to work with (a Music21 object in our case). Once the melody is manageable, we need to process it to be able to perform all the extractions and computations in the module to create and store the data points that will be used in the merging process. The flow of this process can be seen in figure 4.16.

As it was explained before in section 2.1.1, a music composition is divided into measures and the number of notes that these contain is determined by the time signature of that

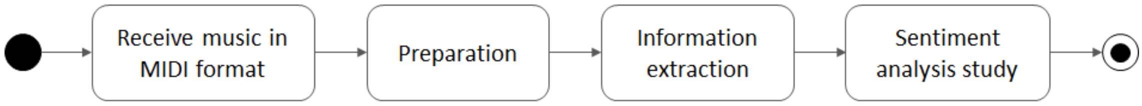

Figure 4.16: Flow of the music information extractor

specific sheet. The use of measures for the evaluation process is determinant as notes by themselves do not contain enough useful information. However, combining a note with the ones surrounding it provides more valuable information for the evaluation. As this division is of so much importance, the first part of the process, just after parsing the MIDI file to a format we are capable to work with, is applying these separators based on the time signature of the composition. These two first steps are taken by means of some Music21 built-in functions that provide exactly the functionality needed, and were done following the process shown in figure 4.17.

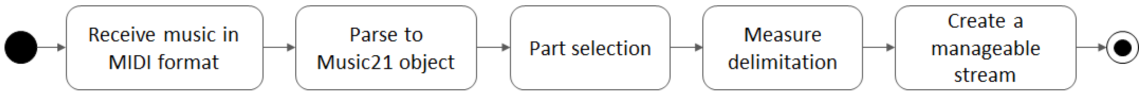

Figure 4.17: Music preparation process

Once the music is structured in a adequate way to work with, it is possible to extract from it more valuable information for the following steps of the process of evaluation.

4.3.3.1. Key

As it was explained in point 2.1.1.2, the key or scale identifies the group of pitches that conform the base in which a composition is written. Depending on the way these pitches are arranged and the tonal distance (defined in section 2.1.1.1) that is between each of them we can determine a set of characteristics that can be associated to the composition. For example, if a song is written in a major scale it tends to be brighter than another one written in a minor scale. This is because the arrangement of the notes in each of them is different and the tonal distance in between notes makes the second one evoke sadder feelings.

Every music composition is written in a key, although not in every music sheet it is explicitly said. The most common key in which a melody is written is *C Major*, and it is the one that most people think of when asked for a scale. Hence, if the key in the composition received by the evaluator module is not detected, we assume it is written in C Major.

Together with the key, it is possible to determine the mode (defined in 2.1.1.3) the composition has been written in. Considering that the order of these modes from happier to sadder is *lydian > ionian (or major) > mixolydian > dorian > aeolian (or minor) > phrygian > locrian*, we can extract a valuable data point for the sentiment analysis with it. If the mode is not specified, or is not included in the list, it was defined that the mode by default is *Major*, because it is the most commonly used and usually not specified.

4.3.3.2. Metronome mark

The metronome mark, or tempo, was defined in 2.1.1.6 as the speed at which the music has to be played. Knowing this can help us make some assumptions about the sentiment analysis evaluation of the music by itself. Although there are more variables involved in the classification of a music composition as sad or happy (using these as the two main feelings that can be evoked to a person), it is reasonable to say that the slower the music is played, the sadder it feels. This is just considering music, as the sentiment analysis of a song with both melody and lyrics is easier to determine, as the lyrics have a more objective meaning that can make a person feel in one way or another.

To obtain this value a built-in function of the Music21 library (see section 3.1.8) is used, as it returns a variable containing the tempo of the composition. However, if the piece being studied does not have a predefined tempo marking, the process marks it as *moderato* by default, as it is the most common and safest one.

The information we have been able to extract up to this point is just a set of overall characteristics of a music composition. These will definitely be valuable assets when determining the merge process' quality, but they do not cover all the information available in the music sheet. Hence, we can deepen the composition and extract some information that can only be found by studying each measure and note.

4.3.3.3. Note length

Analogously to the metronome mark, the length of a note (explained in section 2.1.1.4) can provide us some help to determine if the music is either happy or sad. It is possible to determine that the longer the duration of a note, the slower the music will feel, and the opposite would happen with shorter notes. This is because the briefer a note is, the higher the number of notes it is possible to fit into the measure, and therefore the faster the rhythm.

In a song with lyrics this is not as representative, as the way each syllable is associated to one or more notes is more relevant than the music by itself, but having just the music for the moment, it provides enough information.

4.3.3.4. Pitch

Considering the pitch of a note (defined in section 2.1.1.1) we can also try to determine the degree of happiness it evokes to the listener. A single note by itself it is not very relevant, but if the average pitch of a group of notes is lower it is possible to say that the music is sadder.

For this part it was decided that the pitch was going to be studied based on its MIDI values as it is the most straight forward way to do so with the Music21 library. As MIDI values are directly related to the frequency of the represented note they are reliable, and also as the values range from 0 to 127, the computations with them are more intuitive.

Additionally, if we consider the pitch in relationship to the scale, it is possible to determine another point of value. This is achieved if the note is the *mediant* of the scale (defined in section 2.1.1.2), as it it the most representative or all notes. If the scale is major, we can say that the presence of the *mediant* note brings happiness to the music, and if the scale is in minor mode, it evokes a sadder feeling.

This approach specially is very general as it is a very objective way to decide whether the song is more or less happy, but for a first version of the evaluation it serves its purpose and sets the bases for further improvements.

4.3.3.5. Tendencies

In a music composition, the tendency was defined as the variation of pitch between consecutive notes (see section 2.1.1.1). By observation, it is possible to say that the more descending tendency the melody has, the sadder the feelings it evokes, and the opposite would happen when it is ascending. This is more noticeable if the pitch jumps are consecutive and not abrupt, as a sudden high variation in the pitch is not representative enough of the general feeling the composition evokes.

To extract this information, the process (as shown in figure 4.18) checks every note with its preceding one in each measure. As pitches are easily accessible and are normalized thanks to the MIDI format, both notes' frequencies can be compared. After this comparison, we end up with a value that determines by its sign if the tendency of a measure is ascending or descending (+/-) and by the value the degree to which the measure's pitches vary from each other.

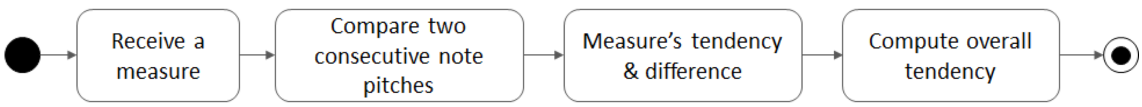

Figure 4.18: Tendency detection process

4.3.3.6. Dynamics

Another method for a composer to communicate the musician the way the composition has to be performed in terms of feelings and interpretations is by setting concrete dynamic articulations (defined in section 2.1.1.7) in different measures or notes. By means of these articulations the composer can tell the performer (or reader) how the music has to be played specially in terms of volume.

The approach followed to treat the information related to these dynamic articulations consists on checking the ones associated to each note and get the volume shift that they produce to its original sound. This shift is represented in a range from -1 to 1 where 0 is the base volume, 1 is the loudest one and -1 the softest.

4.3.3.7. Sentiment Analysis

Having extracted and computed all the previous data points from the music we have designed a process to study the sentiment analysis of a composition. To work with it, it was decided that the sentiment analysis was going to be indicated in different variables, one assigned for each note, one for each measure and another that indicated the overall score for the composition. These variables store a number ranging from -1 to 1, where the lower bound indicated the saddest possible score, and the higher showed the happiest.

To compute the sentiment analysis scores, the process designed uses the previously extracted data points in the following ways:

- **Mode**: the mode, extracted in the process explained in section 4.3.3.1, is the data point that provides the most valuable information in terms of sentiment analysis for the overall composition. The separation between notes in a given mode is the reason why this is achieved because a distance of one tone from one note to the following one is not as directionally strong as two notes separated by half a tone. This strength and the relation between every note in the scale determines in a way the degree of sentiment polarity a composition has.

  The score related to it was set to be in a range from 0 to 1 (see table 4.1), in which the *Locrian* (which is the saddest mode of all the classical ones) is assigned a 0 and the *Lydian* (the brightest of the classical) is assigned a 1. Both the classic denomination and the common one (major and minor) are considered, but as these last two are equivalent to others in the first set, they are scored equally.

- **Metronome and duration**: as exposed in sections 4.3.3.2 and 4.3.3.3, both the

Mode	**lydian**	**ioninan/major**	**mixolydian**	**dorian**	**aeolian/minor**	**phrygian**	**locrian**
Score	1	0.83	0.67	0.50	0.33	0.17	0

Table 4.1: Table with the sentiment analysis scores assigned to each mode

metronome mark in which a music composition is meant to be played and the length of the notes in it serve a valuable purpose to determine the general feeling a song generates. We have identified the metronome mark of the composition, and it is also possible to know the average duration of the notes in it by a simple mathematical computation.

This information by itself is valuable, but both variables can be studied together to provide a normalized approach for the sentiment analysis of the music. For this, a standard was set considering the possible values each of the variables can take.

As seen in section 2.1.1.6, the range of values for the tempo is 0-240, although it can possibly be higher than 240 bpm, but it is taken as the top threshold because it is the last value considered in the classical tempo denominations. On the other hand, the duration for the notes is taken as explained in section 2.1.1.4 and it is measured in quarter. Thus, the crotchet (the most basic note) has a duration of a quarter, the quaver of a half and the minim of two units.

With these values in mind, a standard was defined considering the duration of the note in relation to the tempo. The following table (table 4.2) was created where the values in it are the division of the tempo by the note length. To deal with this table, the values in the diagonal were taken as the representative ones in order to give a score manageable during the merging process (seen in the bottom row of the table). The value obtained after performing the division of the attributes extracted from the music is compared to the highlighted values and assigned the score of the one that is closer to it (the result of their subtraction is smaller). The score assigned ranges from 0 to 1, where the higher the result of the division from the relation, the higher the value will be, and therefore, the happier the music will be considered.

Met / *Dur*	**240**	**225**	**210**	**195**	**180**	**165**	**150**	**135**	**120**	**105**	**90**	**75**	**60**	**45**	**30**	**15**
0.25	960	900	840	780	720	660	600	540	480	420	360	300	240	180	120	60
0.5	480	450	420	390	360	330	300	270	240	210	180	150	120	90	60	30
0.75	320	300	280	260	240	220	200	180	160	140	120	100	80	60	40	20
1	240	225	210	195	180	165	150	135	120	105	90	75	60	45	30	15
1.25	192	180	168	156	144	132	120	108	96	84	72	60	48	36	24	12
1.5	160	150	140	130	120	110	100	90	80	70	60	50	40	30	20	10
1.75	137	129	120	111	103	94	86	77	69	60	51	43	34	26	17	9
2	120	113	105	98	90	83	75	68	60	53	45	38	30	23	15	8
2.25	107	100	93	87	80	73	67	60	53	47	40	33	27	20	13	7
2.5	96	90	84	78	72	66	60	54	48	42	36	30	24	18	12	6
2.75	87	82	76	71	65	60	55	49	44	38	33	27	22	16	11	5
3	80	75	70	65	60	55	50	45	40	35	30	25	20	15	10	5
3.25	74	69	65	60	55	51	46	42	37	32	28	23	18	14	9	5
3.5	69	64	60	56	51	47	43	39	34	30	26	21	17	13	9	4
3.75	64	60	56	52	48	44	40	36	32	28	24	20	16	12	8	4
4,00	60	56	52	49	45	41	38	34	30	26	23	19	15	11	8	4
Score	**1**	**0.92**	**0.83**	**0.75**	**0.69**	**0.63**	**0.56**	**0.50**	**0.44**	**0.38**	**0.31**	**0.25**	**0.19**	**0.13**	**0.06**	**0**

Table 4.2: Table with the relationship between the music's metronome and notes' duration, along with the score assigned for each highlighted value

- **Pitch**: the pitch of a note can also provide valuable information for the sentiment analysis process of the music. As it was explained in sections 2.1.1.1 and 4.3.3.4, the way the evaluator deals with pitches is by using the MIDI numbers relative to their frequencies. In order to assign a score, the pitch value being studied is divided by 127 (the highest it can get due to the way MIDI files work) and thus, we obtain a value in the 0-1 range, consistent with the rest of the scores.

- **Dynamics**: the dynamic articulations associated to a note (defined in section 2.1.1.7) reflect different variations in the way it has to be interpreted and performed, which provides plenty of information for the sentiment analysis. The way the evaluator manages these dynamics is by using the tools that Music21 provides in values ranging from -1 and 1, and normalizing them to a 0-1 range for consistency with the rest of scores.

- **Tendency**: the way the evaluator deals with the tendencies is by checking the information extracted from its study and transforming it into a numerical value. This score computed after the process defined in section 4.3.3.5 considers the tendencies in measures, as they are estimated as the smallest unit for the evaluation for this metric. The value obtained can be either positive if the overall variation of pitches is ascending, negative if it is descending or zero if it is flat (the pitch increases compensate the decreases). To compute the score, this data point is then normalized to a number in the 0-1 range consistent with the other ones.

Having defined these common attributes, the next step moving forward is considering all of them together for the different sentiment analysis scores previously mentioned (per note, per measure and overall) as pictured in figure 4.19. After assigning each variable to every score, they are combined by means of a weighted average in which the mode and the metronome-duration ratio are the most relevant ones, and then the pitches, dynamics and tendencies (if applicable) with the same weight. The flow of the score assignment process for each of the three variables can be seen in figure 4.20.

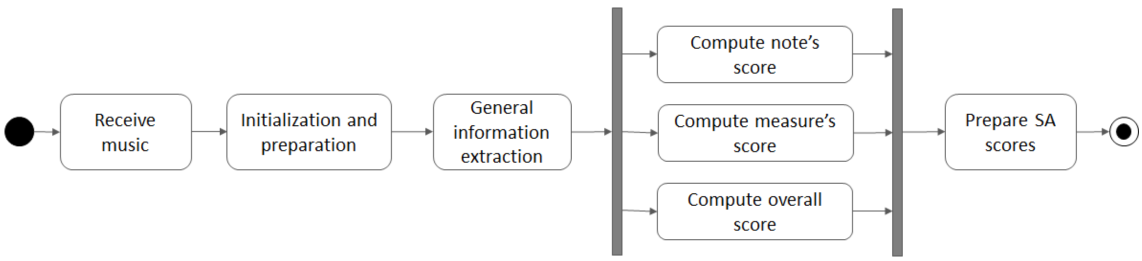

Figure 4.19: Sentiment analysis scores initialization

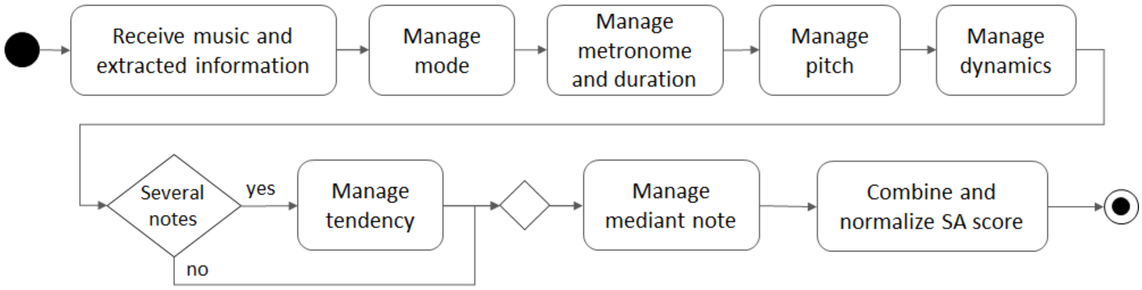

Figure 4.20: Sentiment analysis score preparation

- Score note: the sentiment analysis score for the note is the most basic and the least representative one, as the unit of evaluation is the smallest possible. A single note does not have enough sentiment information by itself, but a set of general rules were defined to approach this matter.

  First of all, the score computed considering the **mode** the music is written in is assigned, as it is one of the main data points known that provides enough information about the sentiment and is applicable to the entire composition. It is also given a score depending on its **duration** and the **tempo** of the song, another one related to its **pitch** and one more due to the **dynamics** that are associated to that single note. Additionally, the process checks if the note is **mediant**, and in that case, it assigns an extra point to the computation if the scale is major, or subtracts it if the scale is in minor mode.

- Score measure: studying the sentiment analysis per measure is probably the best way to do the evaluation of the music as it contains more information than a single note, but it is not as general as the overall score.

  To have a standardized score for the measure's sentiment analysis, all the general variables explained above are applied. The **mode** and the **metronome-duration** ratio (where the duration is the average note length in the given measure) are still the most important ones. In this case the **tendencies** of the music are applicable and are considered when computing the score. Also, the **average pitch** of the notes inside the measure and the different **dynamic articulations** (also considered in average) are used to provide a score. For the study of the **mediant** notes, a counter of these specific notes is computed, and depending on the number of times this specific note appears in the measure, and the mode of the composition, a positive or negative value is added.

- Score overall: for the overall score a similar approach to the one for the measures is taken. This score will also be in the -1, 1 range and it will consider all the variables possible.

As the **mode** of the composition is the same throughout all the melody, the value assigned for it is considered the most important one. For the **metronome-duration** ratio the duration taken is the average of all the notes, as well as for the **pitches**, **dynamics** and **tendencies**, because it is the most reasonable and consistent approach to follow. When considering the number of **mediant** notes that are present in the composition, a rate is computed by dividing how many times this note appears by the total number of notes.

4.3.4. Merging music with poem

After having studied both the music and the lyrics separately, the next and main step in the evaluation process is to merge both and study how well they fit together. To make this process easier and consistent with the different parts of the evaluator module, a similar approach as the one followed for the sentiment analysis study of the music (shown in section 4.3.3.7) was taken, considering a score for the different relevant divisions of both inputs. For the music, scores are given **note by note** and **measure by measure**, and in the case of the lyrics, **verse by verse**. Additionally, an **overall score** is going to be computed considering these three previous scores and other information that cannot be used for any of the divisions.

As a starting point, the general rule considered when combining the music with the lyrics from the poem is that **every note of the melody has to have assigned a single syllable**. This helps provide a rhythm to the song as the way stressed and unstressed syllables are arranged in the poem is how this sensation is perceived. Although not all songs are composed this way, it is the first step for most inexperienced songwriters and, therefore, it will be the first step for the automatic evaluator. Also, as an additional consideration, the **rests** (silences) in the music **are not assigned any lyrics** to make this process as objective as possible. The same happens when two consecutive notes have the same pitch because, as we are working with a piano as the instrument that plays the music, the jump between these notes is not going to be noticeable when played.

For this, the music and the poem are traversed note by note and syllable by syllable respectively, and a series of tests are carried to be able to provide a score for the merging process (see figure 4.21). These scores are computed by increasing or decreasing a value depending on whether a criterion is fulfilled or not, and then normalized to a score in the 0-1 range. However, not every score is modified after every test, as some are not considered to be relevant enough for each one of the scores. In the end, the higher the score is, the better the evaluator considers the merge.

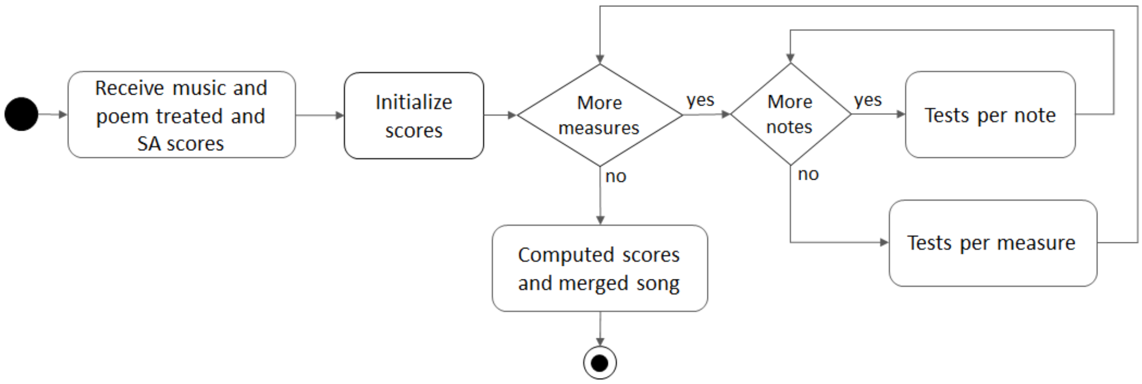

Figure 4.21: Flow of the merge process

All of the following tests will follow a similar approach as the one seen in figure 4.22.

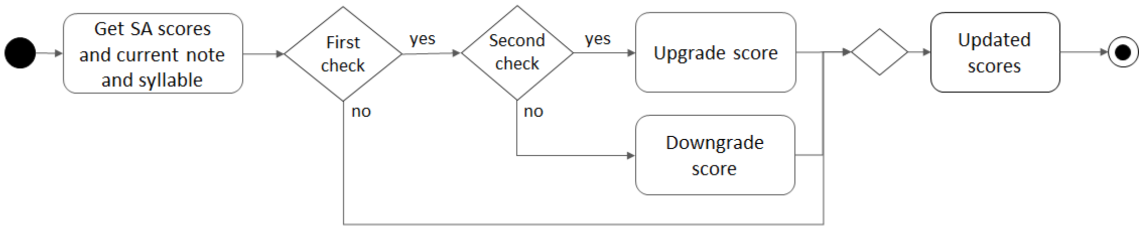

Figure 4.22: General score modification process

4.3.4.1. Starting note of a measure and stressed syllable

As it was explained in section 4.3.2.2, the stress of the syllables in a word is very important as it is what determines the rhythm of the poem. In the case of the music, **the most rhythmically accentuated notes are the first ones in each measure**, because of the division set by the time signature. In non classical music, it is very common to have a drum playing in the background and marking the tempo of the song, with the stronger beats being at the start of the measures.

Considering this, if this first note that marks the beat of the measure has the stressed syllable of a word assigned to it, it should be a sign of quality. On the other hand, if this strong beat is played over an unstressed syllable, it is considered to be a sign of poorer quality. Depending on the result of this checking, the corresponding score is increased or decreased conveniently (see figure 4.22).

The result of this test influences all the scores, as notes, measures and verses are being considered, together with the overall score.

4.3.4.2. Beat strength, importance and stress

Taking a similar approach as in previous test, the syllable is evaluated in terms of its stress and its importance, considering the strength of the beat the current note has. To work with this beat strength, the Music21 library (see section 3.1.8) has an attribute of this same name associated to every note in a composition. This attribute contains a numeric value that indicates the **metrical accent of the note within in the current measure**. This is extracted by means of the time signature, the tempo of the melody and the position the note has inside the measure, and ranges between 0 and 1, where 1 is the most accented a note can be.

To carry out a more complete test, it was decided to consider also the importance assigned to the word. As it was explained in section 4.3.2.1, each word has assigned a degree of importance between 1 and 10, where the lower the number, the more important the word is within the sentence.

The method developed to work with these three variables starts by computing a relation between the beat strength of the note and the importance of the word. To do this, the first is divided by the second, resulting in a positive number smaller or equal to one, and this way we will have a number directly proportional to those two data points. After this, it is checked whether the syllable is stressed or not and, depending on the outcome, different computations are carried out. This is because it was determined that this test could only bring positive results and therefore the scores would never be decreased, even if some test failed. If the syllable is stressed it means that it has to go with a more rhythmically strong note, so the score assigned is higher the stronger the beat of the note is. On the other hand, if the syllable is not stressed, it should be paired with a less rhythmically strong note and therefore the score assigned is higher the lower the beat strength value of the note is. To make this value more significant, the value computed considering the importance of the word and the beat strength is also added. A diagram that shows how this test is performed can be seen in figure 4.23, where "BS" is the note's beat strength and "imp" is the importance of the word.

In this case, all the scores are modified as the four of them are impacted by the results of this test.

4.3.4.3. Last syllable in verse and last note in measure

Considering the definition of the measures done in section 2.1.1.5, we know that their main purpose is to divide and structure the music. Having this in mind together with the previous test defined in section 4.3.4.1, we designed another test to evaluate the quality of the merge.

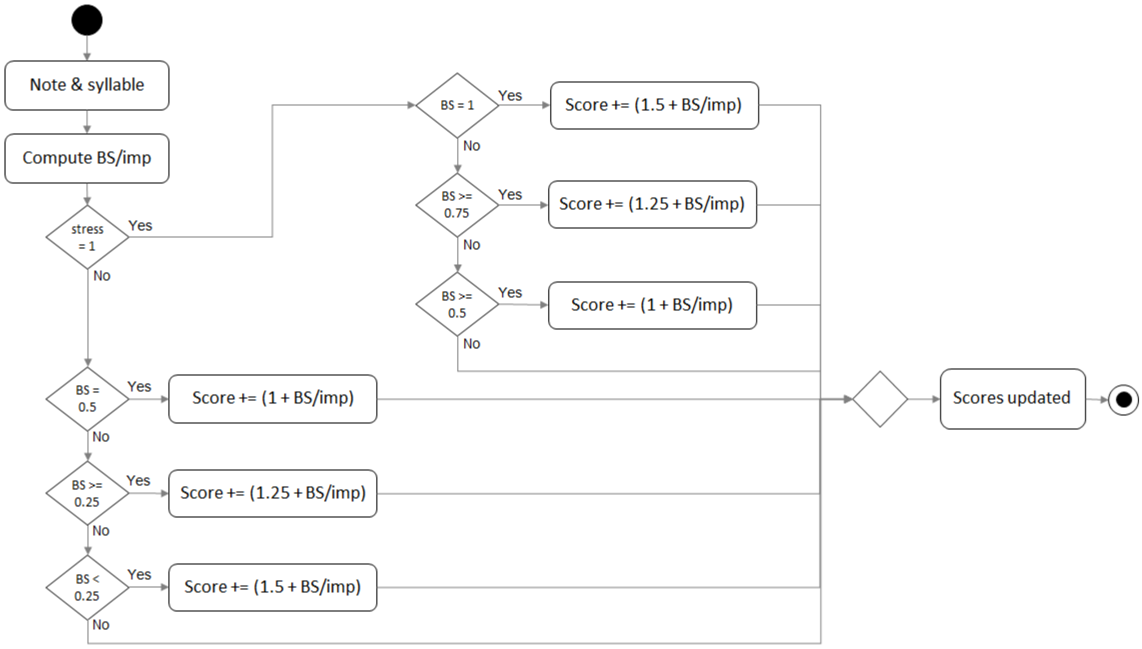

Figure 4.23: Score modification in terms of beat strength, importance and stress

This test checks whether the last note in the measure has the last syllable of a verse assigned to it. This checking is valuable because if it is passed, it would indicate that there is a sense of pause or finish between verses. Additionally, it would implicitly mean that, in most cases, the start of the following verse coincides with the start of a new measure. This is useful considering the low rhythmic strength the last note of a measure has, and the weak stress the last syllable of a verse has out of the entire sentence.

After taking this test, all four of the scores are varied as every one of them is influenced by its outcome.

4.3.4.4. Rest after last syllable in verse

To continue with the checks considering the ending of a verse, a test was defined regarding the rests in the music. This is done by checking the musical figures that appear right after the last syllable of the verse. In this case, if the figure is a rest (or a silence), it would show a clear separation between the current verse and the following one. However, this is a very objective approach, as in poetry and song-writing not all the endings of a verse are followed by a pause, and additionally, there are usually rests between words in the middle of the sentence. We are neither considering lexical endings of the sentences as after the information extraction of the poem, the punctuation marks are removed for the evaluator to work correctly.

After this test, all scores are modified because all of them are affected by the evaluation.

4.3.4.5. Last syllable in verse with tonic note

As it was explained before in the music information extraction in section 4.3.3.4, each note in a scale has a different meaning or provokes a distinct sensation. In this case we are interested in the **tonic** note, the one that determines the starting point from which the scale is built. This note is the **most stable one in the entire scale** and, because of this, melodies usually tend to start and finish with this note.

Knowing the meaning of the tonic note in the current scale it is possible to determine if it fits satisfactorily with the lyrics it has been assigned. As it provides a stability sensation from a musical perspective, it was defined that if the ending of a verse coincides with the tonic then the quality of the merge is higher. On the other hand, if this does not happen, the score is then decreased.

The result of this test influences all scores, as it i considers notes, measures and verses. Also, the overall score is modified.

4.3.4.6. Importance of the word and note duration

Taking into consideration the different duration a note can have as seen in section 2.1.1.4, it was determined that the longer the note is, the more relevant it is in the melody. This can be studied together with the importance a word has inside a sentence given its part of speech (explained in section 4.3.2.1), so a relation was defined that takes both variables' importance. If their values are directly proportional, then the quality scores are conveniently increased.

This test only modifies the scores if the duration of the note that is being analysed is longer than a crotchet and the word class is one of the most important ones in the list. This was the approach taken because it was not considered safe enough to change the score when the word is secondary and the length of the note is short, as it would result in a significant decrease that is not representative.

All four score attributes are affected by the results of this test.

4.3.4.7. Completeness in the order of notes

When the music was studied to extract the sentiment analysis information in section 4.3.3.7 it was mentioned that, depending on the mode the composition was written in, different notes and groups of notes had various meanings. However, this does not only apply to the sentiment analysis, but it is also possible to perform some quality validations comparing consecutive notes.

Due to the separation between consecutive notes in a scale (see figure 2.6), some have a higher tendency to require a specific note to be played after, mainly because the whole tone or semitone separation between them. This is also applicable to chords, although as they are formed by several pitches stacked altogether, they are not as easy to work with. To deal with this problem, the standard defined for the validations of melodies with chords is taking it's root and work with it as if it was a single note.

To make this part of the process as objective as possible every note is studied together with the ones surrounding it. Note that the lyrics are not relevant in this part of the validation as it studies just the completeness of the music itself, checking the objective quality of the composition. The flow of this validation process can be seen in figure 4.24.

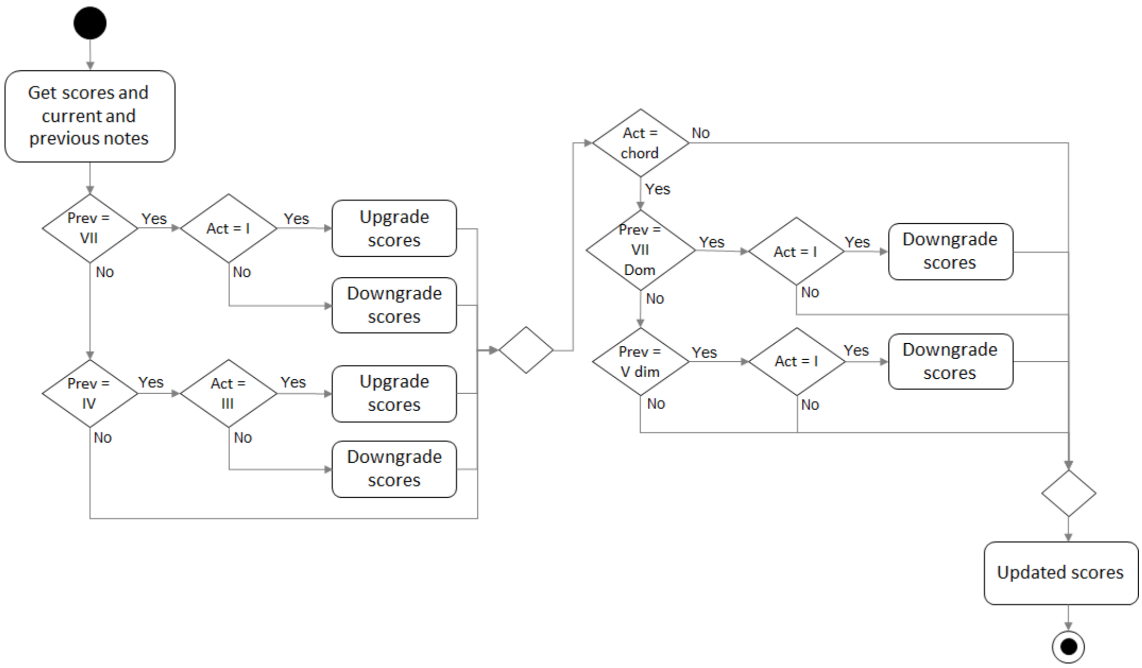

Figure 4.24: Completeness validation process

Some completeness relation checks are the following:

- **Leading tone → Tonic**: as explained in section 2.1.1.4, the *tonic* note in a scale is the one that provides the most stability feeling to the music. On top of that, if we consider that the *leading-tone* in the scale is a semitone below the tonic, we can say that they should be one after the other for the melody to feel complete. This is because of the high directionality the leading tone has towards the tonic (hence its name), specially in a major scale.

  On the other hand, in a minor scale the distance between *leading-tone* and *tonic* is of a whole tone, so the directionality will not be as noticeable. However, as the tonic is still the most stable note in the scale, there would be a slight sense of incompleteness if one is not after the other.

- **Supertonic → Tonic**: keeping in mind that the *tonic* is the most stable note in the scale, it is possible to do another validation regarding this. In the scale, the note that follows the tonic in ascending order is the *supertonic* and, although they are separated by a whole tone, this also makes it require the tonic to be played afterwards to be more polished.

- **Subdominant → Mediant**: taking into consideration the tonal distance between the different pitches of the scale, we can see that not only the *leading-tone* is separated by a semitone to the *tonic*, but the *mediant* and the *subdominant* notes have this same separation. Knowing this, it is also possible to determine that the melody would evoke a more complete feeling if the *mediant* was played after the *subdominant* note.

  The reason the validation checks the pitches in this order and not in the opposite one is because of the importance the *mediant* note has in the scale.

- **Diminished fifth / dominant seventh chord → Tonic chord**: chords also provide this tension feelings to the listener in the melody, so another check was implemented to modify the scores conveniently. In this case, the chords that are more directional in the way they require stability after themselves are the *diminished fifth* and the *dominant seventh* chords, so the validation process checks if they are followed by the *tonic* chord.

  In this case it is the chord what leads to this tension due to the way pitches are arranged to create the specific sound, so the process does not convert them to single notes using the root conversion explained before.

In this case, as the lyrics do not take part in the validations, only the scores relative to the music are changed (i.e. note and measure), as well as the overall punctuation.

4.3.4.8. Sentiment analysis comparison

Regarding the sentiment analysis of both the music (explained in section 4.3.3.7) and the poem (obtained as an input from the composer), both data points are in a consistent format and thus, it is possible to work with them together. These two scores are in a [-1..1] range, where the thresholds identify the saddest and the happiest the parts could be respectively.

To take a normalized approach for the management of the sentiment analysis scores the difference between them is computed. Depending on how high the result is, the quality score will rise more or less. In this case it was decided that the score would never be lowered because both sentiment analysis scoring algorithms are very basic and they are not considered to have enough meaning to do so. This process can be seen in figure 4.25.

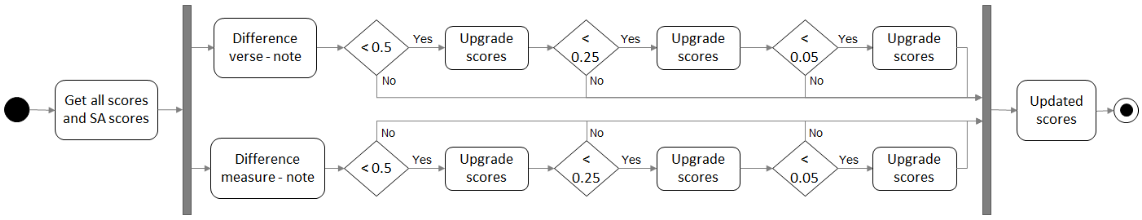

Figure 4.25: SA score comparison process

4.3.4.9. Word's importance and note's pitch relation

As it was mentioned before, in section 4.3.4.2, there is a relationship between the importance of a word and the beat strength of the note associated to it. Taking a similar approach to this, we designed a test that checks the importance of the word together with the pitch of the current note.

The test defined checks if an important word is sung at the same time a high-pitched note is played. This is because, generally, the higher the pitch of a note the more noticeable it is within the melody. Therefore, if the highest pitched note in a measure has been assigned a syllable of the most important word of the verse, the quality score or the merging process is increased.

4.3.5. Output and feedback

The evaluator module has been developed considering that its input information are the poem as a text and the music automatically generated in a MIDI format, as well as the sentiment analysis score of each verse for the first of these two. To make the output of the application as good and consistent as possible, this module sends the following feedback to the rest of the components for them to take it and make the pertinent modifications to their previous output.

- **Worst scoring verse**: as both the poetry generator and the evaluator component can identify the different verses inside the poem that is being treated, the output defined for the evaluation module is the index of the verse that has scored lowest of all after the merge process. This can be easily taken by the poetry generator and then used for the improvement process.

- **Worst scoring measure**: although the music has had two different scores along the evaluation process (per note and per measure), it was decided that the most valuable indicator of these two is the second one as it is affected by more tests and therefore, contains more information. The output from the evaluator component meant to be taken by the music generator is the offset of the lowest scoring measure, because it was the most normalized variable as both parts could easily identify the marker within the composition.

In addition to this feedback sent to the other submodules, the evaluator provides different outputs every time its execution takes place.

- **Generated music sheet**: as the main result obtained from the application, the music sheet with the music and the lyrics associated to it is provided in XML format. With it, the intermediate results can be manually studied and the final result can be checked by the user.
- **Computed scores (per note, measure and verse)**: mainly for the development process to check if the quality of the result obtained is good or not. But also, it is useful for the user in case he/she is interested in the score assigned to the composition generated.
- **Music and poem generated**: the same items as received by the evaluator, in case the user wants to try and find a better way to manually merge them, or just wants to check them separately.

4.3.6. Testing

Once the implementation of the evaluator module was finished, a set of tests were performed to check the quality of the results it provides. First of all, a set of already existing songs are evaluated as if they were obtained from the automatic compositors. These tests are carried because the outcomes expected are supposed to be of good quality, as well as the correct fit of the lyrics on top of the music. With these results, it will be possible to define a sensible score threshold to determine if an automatically composed song is good enough in order to stop modifying it.

4.3.6.1. Initial considerations

Before going into detail about the tests carried out, it is necessary to point some features of the developed evaluator that are undesirable and were considered during the design, but were accepted for this approach.

- At the beginning of many songs there is usually a brief portion where there is no lyrics to be sung, but only music. This serves as an introduction to the song and is often a way in which the listeners are able to identify the song. However, as the number of notes composing this part varies considerably between songs, it was not possible to establish a default starting point for the lyrics to start being assigned. Therefore, this version of the evaluator starts assigning lyrics from the first note of the music.

- It is very common to see that a single syllable is sung during more than one note. This poses no problem if the pitch of the consecutive notes is the same, as the evaluator already deals with this situation satisfactorily. However, if the pitches of these notes are different, the evaluator assigns each of them a different syllable. This first version works this way because it is more probable that consecutive notes with different pitches are meant to be sung with a different syllable, and it was not possible to determine a convenient way to identify these syllables.

- After parsing the MIDI files, it is prevailing to see that the format of the notes change comparing side by side the original and the evaluated music sheets. However, the results are equivalent, as this change only affects to the way the notes are shown but not played. A clear example of this can be seen in figures 4.26 and 4.27.

- It is important to notice that, as the songs used for this testing do not come from the generator modules, the input lyrics do no come with an associated sentiment analysis score. To deal with this, it was decided that the default score will be 0, which means that they are emotionally neutral. This will have a direct impact in the computation of the quality scores after the sentiment analysis comparison (see section 4.3.4.8), but it was the most sensible way to deal with it to test the evaluator module independently from the other modules.

4.3.6.2. Selected songs

The set of songs selected for the initial testing of the evaluator has been decided to contain easier songs where the music is not very complex (no chords, no abrupt pitch or dynamic changes, etc.) like the one a beginner musician would choose to practice learning music. Regarding lyrics, there has not been a very restrictive selection procedure, as long as they are written in English and with a not very complex vocabulary and a moderate use of poetic licences.

A small representative sample of the set of tested music is:

- <u>The Beatles</u>: this British pop group is known for writing very simple music and natural lyrics, with not many poetic licenses. Typically, their songs are easily played on the piano, which is an important feature due to the similarity with the later generated music. Their songs selected for this sample are "**Hey Jude**" and "**Hello, goodbye**".

- Louis Armstrong: he is one of the most important jazz icons from the mid 1900s in America. The song "**What a wonderful world**" was chosen due to its simplicity and the relation of the music with the lyrics and the similarities found between it and the expected generated music.

- Ed Sheeran: in order to test with a more contemporary music, the song "**Thinking out loud**" from this British songwriter was chosen. Similarly as with the previous choices, this work was selected due to its simplicity and the relationship with the generated music the evaluator is going to work with in the future.

4.3.6.3. Problems found and modifications

During the initial implementation of the evaluator module, the quality scores obtained were not normalized. This feature was known but it wasn't initially considered of great importance. However, once the first version was finished and ready for testing, it was found that this property was indeed crucial for the definition of the quality score threshold.

It can be seen in table 4.3 that the results obtained are very different between the tested songs. The first idea was to set a 150 score threshold, but this was later discarded because it was not considered objective enough. It was then noticed that usually, the longer the song, the higher the score it got (all examples in the table but the last one show this tendency). This was reasonable as not all the songs portrayed were the same length, regarding neither the music nor the lyrics, and as it was explained in section 4.3.4, this score was computed by varying it based on the tests passed.

Song	**Hello, goodbye**	**Hey Jude**	**What a wonderful world**	**Thinking out loud**
Overall score	327.6116	235.2856	206.1338	177.3256
# of measures	76	56	37	72
# of notes	286	258	148	360
# of verses	45	32	29	39

Table 4.3: Overall scores and relevant information obtained with the first implementation of the evaluator

To try to solve this problem, it was decided to attempt a normalization of the score in terms of the number of measures the song has. In order to get this, the initial score was divided by the number of measures, obtaining then a score that considered both. However, this was found not to be enough, as the same problem for defining the threshold still existed.

The approach taken to deal with this issue also was considering the length of the song. It was found that, if the scores per measure were normalized, the overall would also be. Similarly, for this score to be standardized it is necessary to determine a way to normalize the score of the notes individually. After all the tests performed during the merging process (see section 4.3.4) it was determined that the highest score a single note could have was 10. Thus, if the obtained score for a note is always divided by 10, we obtain a normalized number for each of them. Having these, it is possible to get a standardized score for each measure by performing a simple arithmetic mean of all note scores.

After all these changes, we finally obtain a normalized score for every evaluated song, and therefore it is possible to determine a threshold for the validity of the result.

Song	**Hello, goodbye**	**Hey Jude**	**What a wonderful world**	**Thinking out loud**
Overall score	0.32497	0.19644	0.24574	0.22420

Table 4.4: Normalized overall scores obtained after the modification of the evaluator

Having both tables (4.3 and 4.4) side by side, it is possible to extract some conclusions regarding the changes performed. Initially, the normalized results were expected to be proportional to the non-normalized ones, but this does not happen. This is because the score is normalized from the note level up, and therefore all the scoring results change. However, these normalized outcomes are more sensible, and thus, they were decided to be the final ones of this version.

4.3.6.4. Results obtained

As we can see in the following figures, the resulting music sheet from the merge process of lyrics and melody is not always the same as the original one. This is mainly because of the features shown in section 4.3.6.1, but also due to the division of syllables the evaluator performs. In some cases it does not split words the same way the songwriter did when composing, and in some other cases punctuation signs and apostrophes make the syllable separation function fail.

It is also noticeable that there are some syllables that in the original music sheet are sung while several notes are played whereas in the result from the evaluation this does not happen (figures 4.26 versus 4.27; 4.28 versus 4.29; 4.30 versus 4.31), which was a known issue from the start of the implementation process. However, it had been decided that it was not a top priority, although it is one of the capabilities a further version of the application should consider improving.

Figure 4.26: Example 1: The Beatles - Hello, Goodbye (original)

Figure 4.27: Example 1: The Beatles - Hello, Goodbye (evaluated)

Figure 4.28: Example 2: The Beatles - Hello, goodbye (original)

Comparing the results obtained after testing with Ed Sheeran's “Thinking out loud” (figure 4.33) with the original music sheet (figure 4.32), some interesting points can be perceived. First, there is a practical example of the issues with the used function for

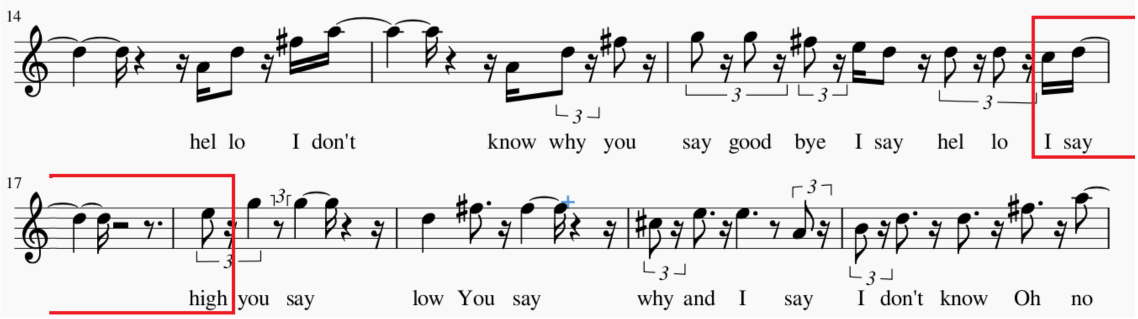

Figure 4.29: Example 2: The Beatles - Hello, goodbye (evaluated)

Figure 4.30: Example 3: The Beatles - Hey Jude (original)

Figure 4.31: Example 3: The Beatles - Hey Jude (evaluated)

splitting a word into syllables as the word "cheeks" is evaluated as a bisyllabic word ("cheek-s"), probably due to the presence of the question mark "?" written after it. However, this syllable-note mismatch is eventually reverted thanks to the assignation of a single syllable to a note when it should have been linked to two of them. This can be seen in the last part of the measures shown, as they match perfectly.

Regarding the scores, it is safe to say that the earlier the mismatch between the original and the generated music sheets occurs, the lower the score will be. This happens because once an "error" is made, it impacts the following parts of the song, and these kind of mismatches are always stacked one after another.

Figure 4.32: Example 4: Ed Sheeran - Thinking out loud (original)

Figure 4.33: Example 4: Ed Sheeran - Thinking out loud (evaluated)

Chapter 5

Sample Execution

In this chapter we will show a real example of the developed solution explained in chapter 4. We will divide this example in two parts, one for each of the submodules that have been described in the previous chapter. First the music and the lyrics are generated, and later they are sent to the evaluator module for it to merge them and determine the quality of the result and send feedback to the generators if necessary.

5.1. Lyrics Module

5.1.1. Valid result

Below we show the whole process that follows the generator in its different phases where you can see the generator of phrases with its analysis of feeling, the final analysis by verse and a suitable final result.

The generator will look for 10 phrases with a minimum length of 50 and a polarity between [[-0.7, -0.25], [0.25, 0.7]] (figure 5.1).

```
[i] Buscando 10 frases con longitud minima de 50 y polaridad en estos
    rangos: [[-0.7, -0.25], [0.25, 0.7]]
[+] [FIRST] [POL: 0.39] I love you whisper soft and true Darling I love
     you, oh, you're gonna do, Should you need my love I must go.
[+] [MATCH] [POL: 0.39] I've got so many things, I've got no time for
    you right now, don't bother me.
[+] [MATCH] [POL: 0.50] There is one love I should never have crossed
    She was a girl who's waiting home for me here so I will try to make
     you mine, girl, Until the end of time.
[+] [MATCH] [POL: 0.33] I'll be there to make you feel Tell me I'm a
    lucky guy.
```

```
[+] [MATCH] [POL: 0.50] Hold me, love me, hold me I wanna be your man,
    I wanna be your man.
[+] [MATCH] [POL: 0.50] But oh well honey don't, honey don't I say you
    love me too.
[i] Found 10 of 75 generated sentences
[i] TOTAL SENTENCES: 10
[+] Generando Estrofa Normal
[-] NOT ENOUGH SENTENCES.
[i] Buscando 10 frases con longitud minima de 50 y polaridad en estos
    rangos: [[-0.7, -0.25], [0.25, 0.7]]
[+] [FIRST] [POL: 0.50] Close your eyes and I'll apologize, If you need
     my love I must go.
```

Listing 5.1: Start of generator

After the generation has finished, it shows us the sentiment analysis of each verse as follow (figure 5.2):

```
[i] Song sentiment tuples (polarity, subjectivity):

[[(0.5, 0.6), (0.5, 0.6), (0.0, 0.0), (0.0, 0.0), (0.0, 0.0), (0.0,
    0.0), (0.3, 0.1)], [(0.0, 0.0), (0.1, 0.4), (-0.3, 0.85), (0.0,
    0.0), (0.0, 0.0), (0.0, 0.0), (0.2857142857142857,
    0.5357142857142857)], [(-0.8, 0.9), (-0.09027777777777779,
    0.20694444444444446), (-0.025, 0.125), (0.0, 0.0), (0.5, 0.6),
    (0.5, 0.6)], [(-0.3, 0.85), (0.0, 0.0), (0.0, 0.0), (0.5, 0.6),
    (0.0, 0.0), (0.5, 0.6)], [(-0.8, 0.9), (-0.09027777777777779,
    0.20694444444444446), (-0.025, 0.125), (0.0, 0.0), (0.5, 0.6),
    (0.5, 0.6)]]
```

Listing 5.2: Sentiment analysis

Before being reviewed by the general evaluator, the lyrics of the song read (figure 5.1):

```
She's got the devil in her heart
No she's an angel sent to me
I'm so sad and lonely, Baby
take a walk and look for her.
Love you like no other
baby, Like no other can,
Love you all the time, girl.

I'm a loser And I'm not what I do if she's
not there, I get shy when They start
to stare, I'm gonna Hold her tight.
Some day you'll find that I have
said or done, Tell me that you
love me too, you love me
baby, Like no other can.

I love you Well how can you say
that love is strong now You've
really got a chip on my own.
Well I love you, baby, and you
ought to know that she would leave
me alone, don't bother me.

You say you'll be mine, you're
gonna say you will love her
more, Till I walk out that door Again.
I'm so sad and lonely, Baby
take a walk and look for her.

I love you Well how can you say
that love is strong now You've
really got a chip on my own.
Well I love you, baby, and you
ought to know that she would leave
me alone, don't bother me.
```

Figure 5.1: Lyric full example

With another example text, we can see how the generator changes the phrase corresponding to the needs of the general evaluator when it passes a verse to change (figure 5.2):

Yes, I know that you love me if I could
get my way, I'd get myself locked
up today, but I can't believe
She's gonna tear your heart apart
Oh her lips are really thrilling
I'll take a walk and look for her.

So please listen to me, if
you please, I've got no time
for you right now, don't bother me.
You say you'll be mine, you're
gonna say You love me too.

Tell me why you cried, and why you
lied to me, if you please, I've
got no time for you right now,
feels so right now, dont't bother me.
If you need my love babe, just
like I know that she would leave
me alone, don't bother me.

If I could see you now, I'd try to make
it shine, there's nothing I won't
do if you need my love I must go.
For I know I hate to leave you
flat. Gonna let you down, and I leave
you flat, because I've told you
before oh, you can't do that.

Tell me why you cried, and why you
lied to me, if you please, I've
got no time for you right now,
feels so right now, don't bother me.
If you need my love babe, just
like I know that she would leave
me alone, don't bother me.

Yes, I know that you love me if I could
get my way, I'd get myself locked
up today, but I can't believe
She's gonna tear your heart apart
Oh her lips are really thrilling
I'll take a walk and look for her.

So please listen to me, if
you please, 've got no time
for you right now, don't bother me.
Went out last night, I didn't stay late,
called her home, had a drink.

Tell me why you cried, and why you
lied to me, if you please, I've
got no time for you right now,
feels so right now, dont't bother me.
If you need my love babe, just
like I know that she would leave
me alone, don't bother me.

If I could see you now, I'd try to make
it shine, there's nothing I won't
do if you need my love I must go.
For I know I hate to leave you
flat. Gonna let you down, and I leave
you flat, because I've told you
before oh, you can't do that.

Tell me why you cried, and why you
lied to me, if you please, I've
got no time for you right now,
feels so right now, don't bother me.
If you need my love babe, just
like I know that she would leave
me alone, don't bother me.

Figure 5.2: Changed verse after evaluation

The left column is the original song and the right column the song once revised. The evaluator ask us to change the 10th verse so, as that verse wasn't a full phrase by itself, we will also change the next verse.

5.1.2. Invalid results

With different sentiment analysis configurations we can see that, in figure 5.3 most of the verses are repeated as we configure the sentiment with the values: [-1, 0,5], [0, 0,5]. In figure 5.4 the sentences are more random between them and don't have much coherence with values [[0, 0.3], [0, 1]].

```
She's got the devil in her heart
Oh, no, no, no this I can't so I will try
to show that I'm not trying to pretend.
She thinks of him, And though he'll never
come back, she's dressed in black.

Whoa, oh, I want no one to talk
to people that I have gone.
Every day we'll be happy, I know,
Now I know love will never die.
Every day we'll be happy, I know,
Now I know love will never die.

I don't care, There's no fun in
what I do is hang my head and moan.
Every day we'll be happy, I know,
Now I know love will never die.
Oh, you're giving me the same
but I'm to blame, it's plain to see.

Whoa, oh, I want to spend Another day
here Oh, oh, oh, You can't do that.
You say you'll be mine, you're
gonna say You love me Too.
She's got the devil in her heart
Oh, no, no, this I can't so I cry instead.

I don't care, There's no fun in
what I do is hang my head and moan.
Every day we'll be happy, I know,
Now I know love will never die.
Oh, you're giving me the same
but I'm to blame, it's plain to see.
```

Figure 5.3: Song example with sentiment values near -1

```
You say you'll be mine, you're
gonna say You love me Too.
Some day you'll find that I love
you, oh, you're gonna say you
will when you won't, oh honey, don't.

Well sometimes I love you
and all I want no one to talk
to people that I have to go.
You love me too, hoo, hoo, hoo,
hoo, oh, And when I, I want you around, yeah.

Well I gave you everything
I had, But you left me I'm a loser
And I lost someone who's near to me
She's got the devil in her heart
No, no, not a second time.

Hold me, love me, hold me, hold
me, hold me, hold me, love me.
I've got a hold on me, baby
I love you and all I gotta do,
Is call you on a Saturday night,
Sunday morning you don't look right.

Well I gave you everything
I had, But you left me I'm a loser
And I lost someone who's near to me
She's got the devil in her heart
No, no, not a second time.
```

Figure 5.4: Song example with sentiment values near 0

5.2. Evaluator module

Having generated the music and the lyrics, they are both sent to the evaluation module for their merging and review. The flow of the execution is the one defined in section 4.3, first treating the lyrics and the music separately, and then combining them and evaluating the result obtained before providing the output and feedback.

5.2.1. Lyrics processing

From the lyrics generation module the evaluator receives the lyrics as plain text (figure 5.5), and also the sentiment analysis scores associated to each of its verses (figure 5.6).

Note that the sentiment analysis is received in a list format. This list contains at

```
You say you'll be mine, you're
gonna say You love me too.
So please listen to me, if
you please, I've got no time
for you right now, feels so right now.

And when I, I want to leave you, you know
the things she does, She does for me,
Whenever you want me at all.
And then while I'm gone please let
me through, I've got a whole lot
of things to do, than to break my
Heart again, this time I will know.

You say you'll be mine, you're
gonna say you will love her
more, Till I walk out that door Again.
Oh, you're giving me the same
but I'm to blame, it's plain to see.

I wanna go but I can't so I cry I'm a
loser And I'm not what I appear to be
mine, you're gonna do, Should
you need a love that's true, It's me.
I've got a hold on me I want you
to do Is just hold me, love me.

You say you'll be mine, you're
gonna say you will love her
more, Till I walk out that door Again.
Oh, you're giving me the same
but I'm to blame, it's plain to see.
```

Figure 5.5: Received song text

```
[[(0.0, 0.0),
(0.5, 0.6),
(0.0, 0.0),
(0.0, 0.0),
(0.2857, 0.5357)],

[(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.2, 0.4),
(0.0, 0.0),
(0.0, 0.0)],

[(0.0, 0.0),
(0.5, 0.6),
(0.5, 0.5),
(0.0, 0.125),
(-0.2143, 0.3571)],

[(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.425, 0.625),
(0.0, 0.0),
(0.5, 0.6)],

[(0.0, 0.0),
(0.5, 0.6),
(0.5, 0.5),
(0.0, 0.125),
(-0.21429, 0.3571)]]
```

Figure 5.6: Received SA scores

the same time one list per stanza, whose elements are tuples of numbers where the only relevant information for the moment is the first element. Thus, this list is converted to a single list containing only the valid data, resulting in the list seen in figure 5.3

The first step of the lyrics' treatment is transforming the inputs into a format more manageable for further operations as seen in section 4.3.2. After this, the resulting structure contains a list for every verse of the lyrics and an index identifying it's position. This list is then processed to determine the part of speech and importance of every word (explained

```
[0.0, 0.5, 0.0, 0.0, 0.2857, 0.0, 0.0, 0.0, 0.0, 0.2, 0.0, 0.0, 0.0,
    0.5, 0.5, 0.0, −0.2143, 0.0, 0.0, 0.0, 0.425, 0.0, 0.5, 0.0, 0.5,
    0.5, 0.0, −0.2143]
```

Listing 5.3: Sentiment Analysis list processed

in section 4.3.2.1) and the last step is dividing each word into syllables and identifying the stressed one (see section 4.3.2.2).

The resulting structure is a list of dictionaries, where each of these contains the information for a single syllable. These dictionaries have the syllable itself as the key and the value is another dictionary containing its relevant information. A sample of this data structure can be seen in the extracted code shown in 5.4.

5.2.2. Music processing

With the lyrics part ready to be merged, the next step is processing the music received. The input music is in MIDI format, so the first step is parsing it to a manageable stream for the further operations to be performed. The music sheet of this file can be seen in figure 5.7, and the obtained stream that will be used for the later processes in 5.5.

Once this stream is ready, the overall information and the data for the sentiment analysis are extracted and computed. To obtain the sentiment analysis scores, all these data points are put together and a weighted average is computed considering the weights exposed in figure 5.6. In figure 5.7 we can see some data points computed using the functions defined along section 4.3.3. The data points per measure and note are also extracted and computed in a similar way, and later combined and assigned to their corresponding variable. All the resulting sentiment analysis score structures are shown in figure 5.8 (note that in the scores for notes and measures, the keys are the offset where the actual note/measure starts).

5.2.3. Merging the lyrics and the music

Having both inputs processed and all the data points extracted, the next step is assigning the lyrics to the music. During this process, all tests shown in section 4.3.4 are performed and the quality scores are conveniently computed. The resulting music sheet, along with the scores obtained can be seen in figures 5.8 and 5.9 respectively.

This, together with the file in MIDI format, would be the final output of the application.

```
[   {'You': {'stress': 1,
             'part_of_speech': 'pronoun',
             'importance': 6,
             'verse': 0}},
    {'say': {'stress': 1,
             'part_of_speech': 'verb',
             'importance': 1,
             'verse': 0}},
    {"you'll": {'stress': 1,
             'part_of_speech': 'noun',
             'importance': 2,
             'verse': 0}},
    {'be': {'stress': 1,
             'part_of_speech': 'verb',
             'importance': 1,
             'verse': 0}},
    {'mine': {'stress': 1,
             'part_of_speech': 'noun',
             'importance': 2,
             'verse': 0}},
    {"you're": {'stress': 1,
             'part_of_speech': 'noun',
             'importance': 2,
             'verse': 0}},
    .
    .
    {'So': {'stress': 1,
             'part_of_speech': 'adverb',
             'importance': 4,
             'verse': 2}},
    {'please': {'stress': 1,
             'part_of_speech': 'noun',
             'importance': 2,
             'verse': 2}},
    {'lis': {'stress': 1,
             'part_of_speech': 'noun',
             'importance': 2,
             'verse': 2}},
    {'ten': {'stress': 0,
             'part_of_speech': 'noun',
             'importance': 2,
             'verse': 2}},
    {'to': {'stress': 1,
             'part_of_speech': 'other',
             'importance': 10,
             'verse': 2}},
    {'me': {'stress': 1,
             'part_of_speech': 'pronoun',
             'importance': 6,
             'verse': 2}},
    {'if': {'stress': 1,
             'part_of_speech': 'preposition',
             'importance': 5,
             'verse': 2}},
    .
    .
```

Listing 5.4: Data structure after processing the lyrics

```
{0.0} <music21.stream.Measure 1 offset=0.0>
   {0.0} <music21.clef.TrebleClef>
   {0.0} <music21.meter.TimeSignature 4/4>
   {0.0} <music21.note.Note G>
   {3.6667} <music21.note.Note D>
{4.0} <music21.stream.Measure 2 offset=4.0>
   {0.6667} <music21.note.Note C>
   {2.1667} <music21.note.Rest rest>
   {2.25} <music21.note.Note E>
   {3.25} <music21.note.Note A>
   {3.75} <music21.note.Note C>
{8.0} <music21.stream.Measure 3 offset=8.0>
   {0.75} <music21.note.Note A>
   {1.25} <music21.note.Rest rest>
   {1.3333} <music21.note.Note E>
   {1.8333} <music21.note.Rest rest>
   {2.0} <music21.note.Note E>
{12.0} <music21.stream.Measure 4 offset=12.0>
   {0.5} <music21.note.Note G>
.
.
```

Listing 5.5: Stream obtained after processing the input music

```
weights_music = [3, 3, 1, 1, 1, 1] # mode, met/dur, pitch, dynamics, +[
    tendency], mediant
weights_measure = [3, 3, 2, 1, 1, 1] # mode, met/dur, pitch, dynamics,
    +[tendency], mediant
weights_note = [3, 3, 1, 2, 2] # mode, met/dur, pitch, dynamics,
    mediant
```

Listing 5.6: Weights for the computation of the sentiment analysis scores of the music

```
metronome_mark = 92
key = <music21.key.Key of C major>

SA_mode = 0.83
SA_metronome_duration = 0.438
SA_by_pitch_music = 0.49606299212598426
SA_dynamics_music = 0.33847117794486214
SA_tendency_music = 18.8125
```

Listing 5.7: Overall information extracted from the music

Figure 5.7: Received song's music sheet

However, as the overall score obtained is not considered good enough (it its below 0.2, the determined valid threshold), this module sends feedback to the generators in order to try and obtain a better result (see section 4.3.5). This feedback consists on identifying the worst scoring verse and measure and notifying the poem and the music generators respectively of their positions.

After this feedback is sent, the evaluator module waits until the generator modules produce the new inputs. With this updated input data, the process starts at the starting point of section 5.2.

5.2.4. Interpretation of the results

With this first implementation of the evaluator module, the results obtained (see figures 5.8 and 5.9) are considered to fulfil the initial expectations. As we can see, the music sheet

```
SA_note =     {0.0: −0.1503507516105942,
              3.6666666666666665: 0.06475733715103771,
              4.666666666666667: −0.02346743020758768,
              6.25: 0.4312569792412311,
              7.25: 0.09142161775232638,
              7.75: 0.0618940586972081,
              8.75: 0.09142161775232638,
              9.333333333333334: 0.48225912670007176,
              10.0: 0.2760773085182533,
              12.5: −0.1503507516105942,
              ...}

SA_measure =     {0.0: 0.017796916010498487,
                 4.0: 0.20070042949176803,
                 8.0: 0.23031242543545694,
                 12.0: −0.11706513958482467,
                 16.0: 0.06263224767358633,
                 20.0: 0.2857054998806967,
                 24.0: 0.14525739680267247,
                 ...}

SA_overall = 3.73620683401417
```

Listing 5.8: Sentiment analysis data structures

generated poses a fair example of what the application was made to do, and the scores obtained are also sensible.

As this song is an original (it does not come from another already existing song or is a version of it), some of the problems found during the testing process (section 4.3.6) are not applicable to this result. However, having those present it is possible to identify some features of the song that can be improvable.

The first thing that is perceived when trying to sing the lyrics along with the music is that the duration of the notes makes it difficult in some cases. A clear example can be seen in the sixth measure, where longer word like "please", which is monosyllabic, is assigned to a quaver note. As the evaluator module is not in charge of modifying neither the music nor the lyrics, this could be taken into consideration for the score computation. A possible approach could be similar to the one in section 4.3.4.2 where the beat strength of the note is compared to the semantic importance of the word it is being assigned, but changing the first by the note duration and the second by the number of letters the syllable has. However, not only the number of letters is relevant as we should rely on the phonemes, which sometimes vary among accents (as explained in section 4.3.2), but this could be studied in a further implementation.

In the fifteenth measure we can also see an example of a bad fit of lyric with note.

```
score_verse =   {0: 24.000000000000004,
                1: 8.416666666666668,
                2: 9.564583333333335,
                3: 16.333333333333332,
                4: 23.083333333333336,
                5: 32.5,
                6: 17.666666666666666,
                7: 18.427083333333332,
                8: 20.791666666666664,
                9: 18.366666666666667,
                10: 25.666666666666668,
                ...}

score_note =    {0.0: 0.3666666666666667,
                3.6666666666666665: 0.1,
                4.666666666666667: 0.3,
                6.25: 0.03333333333333333,
                7.25: 0.1,
                7.75: 0.2,
                8.75: 0.03333333333333333,
                9.333333333333334: 0.2,
                10.0: 0.20833333333333331,
                12.5: 0.1,
                ...}

score_measure =   {0.0: 0.23333333333333334,
                  4.0: 0.15833333333333333,
                  8.0: 0.14722222222222223,
                  12.0: 0.1,
                  16.0: 0.027777777777777778,
                  20.0: 0.11729166666666666,
                  24.0: 0.15733333333333333,
                  28.0: 0.15000000000000002,
                  32.0: 0.1,
                  36.0: 0.2111111111111111,
                  ...}

score_overall = 0.16412436038011693
```

Listing 5.9: Resulting score lists from the merging process

```
worst_scoring_verse = 1          # score_verse[1] = 8.416666666666668

worst_scoring_measure = 96.0   # score_measure[96.0] =
    0.024999999999999998
```

Listing 5.10: Worst scoring verse and measure

Figure 5.8: Resulting song's music sheet

In this case, the word “the”, a determiner, one of the lowest important word classes, is assigned to a minim, which is one of the longest lasting notes. This is noticeable when reading the music sheet, and also when checking the scores obtained, as the one computed for this note and measure is of **0.1**, whereas the surrounding measures all score higher than **0.3**.

Similarly, the worst scoring measure (figure 5.9) is another example of a low importance word assigned to a long lasting note. Apart from this, the fact that all the words in it are not very relevant (considering the ranking defined in 4.3.2.1), along with the fact that the long note is a G (dominant in this scale) and it is associated to “lot”, make it possible to understand why this result was obtained.

Additionally, regarding the scores assigned to the different verses (see figure 5.9), we

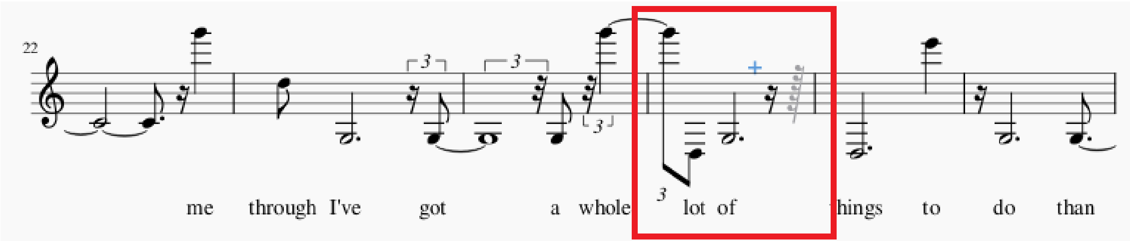

Figure 5.9: Worst scoring measure

can see that they are not normalized as the ones for the music are. This is not an important problem, as the way they are computed is more standard than the original one used for the music. However, a similar approach can be taken to achieve a higher standardization of the scores, like dividing the score by the number of syllables of the verse (although usually all verses would have the same amount) or computing a weighted average considering the importance of the words.

Chapter 6

Contributions

During the development of this work we all actively participated in each of the phases. All members' involvement degree was high and when problems came, we managed to solve together as a team. Although the work done was a huge challenge for all reasons above mentioned, we all participated as a team when tough times came.

6.1. Preamble

As it was mentioned before, for the development of our solution we have used the song generator designed and implemented by Guillermo Villeta Torres. This module composes a musical piece automatically and provides it as an output for the evaluator module to use it, and also is capable of receiving the feedback from this evaluation and modify the music conveniently. However, this module does not fall within the scope of this document, and will be explained in a future TFG.

6.2. María Luisa Quiroga

6.2.1. Background

Before I started this project, my knowledge of artificial intelligence was very limited. During my period in the degree I did not study any artificial intelligence subject, however, as a pending branch of knowledge, I found it interesting to study it freely during the development of this work.

On the other hand, I had not programmed in Python before starting this project. During the first six months of the course, I studied "nformation Management on the Web" which allowed me to acquire a base while I started to perform the first tests in Python language. Although at the beginning of the project there was a debate between making the project in other languages such as Java, finally we agreed with the tutors to make this in Python.

Apart from the technical knowledge necessary for the development of this project, there is a very powerful theoretical part related to music and poetry. Personally, I was very familiar with the field of poetry because for a long time it was one of my hobbies. Understanding the basic concepts, the creative and artistic process, and the rules that already existed to be able to adapt them to our project, has been something fundamental to develop the lyrics module more effectively.

6.2.2. Contribution

From the beginning of the project, it was clearly divided into the three main modules: lyrics, music and external evaluator. My main contribution was the development of the lyrics module. I started researching together with my colleague Carlos Martín, focusing on text generators. One of the main works that guided us when we started to investigate and go further was the PoeTryMe application project.

After studying and evaluating all the options we found (see 2.2.2), I started to develop this module based mainly on Markov Chains (see 2.2.2.2) for text generation. I also studied and developed the adaptation of texts to poetry, based on my own experience and using simple structures and rhymes. At this point, we agreed with our tutors that the lyrics would be given priority over the music, because of the flexibility of the latter as it can be allowed to lengthen or shorten the notes to suit the lyrics of the song. After that, although a first option was the use of fitness functions for the adaptation of this text (see 2.2.1.2), these were incompatible to some extent with the final development because of the limitations it presented.

As for the emotional analysis of the texts, my colleague Carlos Martín and I shared experience with different libraries and technologies that performed this analysis and the use of TextBlob (see 3.1.5) as a tool for this work was an almost immediate decision.

The syntactic evaluation for the coherence of the texts has been the greatest challenge and the part that has changed the most. Initially, I relied on different methodologies that I had read during the research at the beginning of the project, however, I ended up developing my own simpler methodology (see 4.2.2.1) that would ensure adequate results.

Finally, I adapted the module to the requirements of my colleague Carlos Martín's external evaluator, with whom I developed the communication between these two modules.

6.3. Carlos Martín Testillano

6.3.1. Background

The main challenge that I faced during the development of this project was the fact that I had to work actively with both music and lyrics, so I had to deepen my knowledge in these fields.

Regarding music, I lacked plenty of technical knowledge because, although I studied music in the past, it was more focused on playing than composing. However, I already knew how to read a basic music sheet, but I did not know how one was composed and how many factors have to be taken into consideration when doing so. On the other hand, I had more awareness on what comes in to writing poetry, but had never tried to use a poem as the lyrics of a song.

Considering technologies, I had little experience with artificial intelligence, mainly using neural networks and different advanced search algorithms. Regarding programming, I had plenty of experience designing and developing algorithms to carry concrete tasks, and also to optimize them in terms of resources and time. However, I had never heard of the concept of *Computational Creativity*, although I knew there were many people working on this field.

Finally, as the programming language decided for the development of the solution was Python (see section 3.1), I had to train on this specific language. I had no experience with this programming language before, so I started taking an course and doing research to avoid any technical difficulties I may face during the implementation.

6.3.2. Contribution

Since the beginning of the design of this project, the idea was to divide the entire solution into three different and independent parts that could communicate with each other. This way, the module I was assigned to define and develop was the evaluator module (see section 4.3), which made it the main communication part in the solution as it is the one that combines what the generators produce.

First of all, I had to investigate all the related work that has been done in the field of computational creativity regarding both music and poetry, keeping in touch consistently with my colleague María Luisa Quiroga with this last part. During this investigation, I collected many ideas regarding the automatic composition of the two of them, considering techniques, technologies and approaches, and shared them with my colleague. However, as the focus of my work was to develop a module that combined both music and lyrics and not the generation itself, I had to look for external help on how these two can be put together. For this, I lightly trained in song-writing in order to acquire the basic abilities to define the module that would later do this work automatically.

Once I considered I had enough knowledge about song-writing and Python programming, I started defining the Evaluator module. My first step was designing the Lyrics information extractor (see section 4.3.2), in which the main issue I faced was defining the data structure to store the information and how to deal with it. This was mainly due to my limited knowledge of the language, but after some research and trials, I came up with a satisfactory solution. To develop these functions related to the poem I required some help from María Luisa, as she was developing the lyrics generator.

When working with music (see section 4.3.3), the first problem I found was the library or module I would work with. I started trying different modules like Mingus or ABC4J, but they did not provide us with enough functionalities. However, when I discovered Music21 (see section 3.1.8), I found out I could not only carry the operations I had in mind during the design, but more advanced ones.

After having both information extractors, I defined a series of tests to be carried during the merging of the music with the lyrics (see section 4.3.4). The definition of these checks was one of the hardest parts of the design process, as it involved a deep knowledge of song-writing which I initially lacked, but I think that the design obtained meets satisfactorily the specification of the solution. During the testing process, as it was mentioned in section 4.3.6, I found out that the results I was getting were not normalized, so I had to modify the solution. Additionally, I had to find a corpus of songs that could help me define the solution and also test it.

Also, during all the project, I had to be in touch with both of my colleagues because I wanted to have some feedback from them and also to define the way the modules would connect with each other. It was challenging to determine what output my module would give, and also which variables would be sent to the other modules. We initially thought of telling the generators the exact word or note in which the quality scores obtained were worse, but it was later discarded because we believed it was not sensible enough. The solution finally was determined to send the worst scoring measure and verse to the music and lyrics generators respectively (see section 4.3.5).

Chapter 7

Conclusions and future work

Computational creativity is a field that has been evolving for many years and there are areas that have not yet been thoroughly studied. Specifically, although the generation of music and poetry has been the focus point of many researchers, there are not many studies about joining them to compose a completely automatically generated song. Having this in mind, we have created our own approach that adds to this universe and will serve for future researchers in this field as a starting point. Although the results obtained are not optimal, we have developed a series of independent modules that serve to this purpose of composing a song automatically which will be open for future improvements or serve as a inspiration for whoever comes after.

Our solution is composed by two generator modules, one for music and another for the lyrics, and a separate module in charge of merging them and evaluating the result. However, due to some complications during the documentation process, the music generation module developed by Guillermo Villeta Torres is not explained in this document and will be separately described in a future TFG.

7.1. Lyrics module

As a result of our work in this module, we have achieved the creation of the lyrics of a song, in verse format, a syntactic analysis that shows coherence between subject and predicate, and a feeling analysis that provides semantic homogeneity throughout the song. Even with this, there are many points of improvement, both for each module and in the general process followed, which we will develop below.

Syntactic analysis On the one hand, we have succeeded in having the *syntactic analysis* module verify the coherence by checking the number of subjects and verbs in the sentence.

To begin with, this approach is well under way, however, it could be improved through the use, or development if necessary, of better text analysers that allows its structure to be identified more effectively. Once the analysis is done, some rules could be pre-established to check that the text created is correctly structured. Among these rules, include some that allow the introduction of literary figures that alter the order of the text, as the hyperbaton, would allow to give a greater level of poeticity to the text. Another possibility would be the introduction of neural networks which, added to the predefined rules, can bring the possibility of learning certain literary figures and incorporating them into texts.

With these proposed improvements, this module could be used as a tool to support teachers and students in the syntactic analysis of sentences in their classes.

Rhyme and structure On the other hand, we don't focus on getting a perfect *rhyme* because in lyrics it doesn't carry as much weight as poems. Apart from that, our approach to structuring the lyrics has been quite successful as is easy to modify to the developer's liking and to establish the structure that best suits what it is needed. To improve the rhyme, you can modify the generation by Markov Chains of sentences, including a condition that the last word must rhyme with last of the first verse generated. You can also include certain rhyme patterns (ABBA, ABAB) depending on how you want the text to rhyme. All of this would serve as a help to writers or as a more entertaining way for students to learn about poetry.

Sentiment analysis Regarding sentiment analysis, the approach of creating the text by sentences and not in an "infinite" loop has allowed us to have more control as we are checking the emptions of each phrase and not in the general text. To improve this part, we believe that the importance does not reside in the way the text is evaluated but in the evaluation itself. This specifically regarding the way the evaluator checks if a word or phrase in one context expresses happiness or sadness in other context. Also, as there exists many different feelings and emotions, trying to develop this branch to get an "emotion analyser" (happy, sad, angry, funny, etc.) could improve the relation between a music and its lyrics. This could be done by training a neural network to which the emotion or emotions of a song's lyrics are specified, and this one analyses them until being able to identify its different emotions.

In addition, this analysis could be taken to the point of being able to identify or associate the lyrics of a song with a particular group, singer or music genre. In this case, the neural network would be trained with corpus of songs by group, singer, or genre, respectively. This could also be used to help existing "song identifiers" or "music playback applications" to find artists and similar songs that appeal to the listener and show him new discoveries.

Other languages Finally, developing this module of song lyrics generator in other languages would bring different perspectives and enrich the project in general. In this case, the most affected part would be the syntactic analysis due to the differences between languages coming from different roots, such as Spanish, English or Chinese.

7.2. Evaluator

In the field of computational creativity, the automatic composition of songs by combining melody and lyrics independently generated is something that has barely been considered. We have seen that there have been several approaches to develop a music compositor and a poetry generator using different techniques, but the work of putting these two together has always been made by a person. This was initially an inconvenient due to the originality of the idea and the lack of previous work to base the development on, but also leaves a very wide margin for improvement.

In the evaluation part, we have achieved to develop an application capable of merging a given set of music and lyrics and detecting whether they fit or not. This is a result of an iterative process where the sentiment analysis of both parts plays a big role when determining the degree of quality of the merge. Along with this analysis, both the music and the lyrics are studied independently in order to extract various valuable data points that will help modify the quality score of the resulting song.

The procedure followed (explained in section 4.3) tries to mimic the one taken by a person when he or she composes a song. As it is one of the first approaches in this field, it could be comparable to an amateur song-writer, who follows a set of simple and very theoretical rules to determine if the resulting song is good enough.

However, as we have mentioned throughout the development, we came up with some ways the evaluator can be improved.

Implement artificial intelligence or deep learning techniques Although we believe that the tests carried are defining enough, by using a non iterative approach based on artificial intelligence or deep learning to determine if the lyrics fit for the melody, the quality of the results would probably be improved. One possibility would be using a group's songs as the dataset for both the music and the poetry generators, as well as for the evaluator. This way, if implemented correctly, the results of the application would be songs that sound extremely similar to the ones already composed by the chosen group or possible versions of the different songs, using composing techniques taken from another song (or maybe another artist).

Improve tendency study The approach followed in this first version of the evaluator for the study of the tendencies in the music to determine the sentiment analysis just considers two consecutive notes. However, in order to have a more conclusive outcome, it would be interesting if this test was taken one step further by making bigger groups of notes, not only in measures but also between them. Another way to possibly enrich this tendency study is considering the degree to which the jump between pitches is made, giving more importance to more distant notes and vice versa.

Define a more complete criteria for the syllable-note matching Taking a look at many popular and more complex songs, it is easy to find cases where there are various notes or chords at the beginning of the composition that are played alone, with no lyrics sang along them. Having a system that can determine when is the best moment in the melody for the lyrics to start would be very valuable and would mean a enormous step towards the "humanization" of the evaluation and merging process. Additionally, if after applying this criteria it was possible to decide if a syllable should be sung during more than one different note, the results obtained would see a big improvement. This could be done using some machine learning or artificial intelligence techniques, as training the module with different songs can help determine the optimum stating point and the words that could be elongated.

Improving the score computation As seen in sections 4.3.6 and 5.2.4, some criteria used for the computation of the quality scores can be added or improved. Comparing the syllable length, preferably phonetically, with the length of the note would help achieving a more determinant score. It would also be valuable to use this syllable length for the assignation of it to several note pitches, as mentioned in the previous point.

Expand feedback information If done altogether with the music and lyrics generators, it would be interesting if a more complete feedback could be sent to them by the evaluator. Among the information that could be valuable for these modules, if it was possible to determine whether a verse is bad because of the stressed syllables, the distribution of the words or the overall meaning and the generator could use this information, the results could see a big improvement. Analogously, being able to communicate the music generator that a measure is not good because its tendency, completeness or sentiment analysis score, would lead the application to better results, and probably improve the number of iterations needed to get to this result.

Decide which module has a better chance to improve the result Similarly to the point above, finding a way to determine which generator's modification would provide a better result after the feedback would mean a big improvement. This decision could be made considering the lowest score of all, as long as they are normalized and comparable. Being able to do this, the execution time would decrease significantly as only one module would have to redo its work and also the result would probably be better after less iterations.

Chapter 7

Conclusiones y trabajo futuro

La creatividad computacional es un campo que ha ido evolucionando durante muchos años y en el que hay áreas que aún no han sido completamente desarrolladas. En concreto, y aunque la generación de música y poesía ha sido el foco de muchos investigadores, no hay muchos estudios acerca de la unión de ambos para la composición de una canción de forma completamente automática. Teniendo esto en cuenta, hemos creado nuestro propio enfoque que se suma a este universo y que servirá de punto de partida para los futuros investigadores en este campo. Aunque los resultados obtenidos no son óptimos, hemos desarrollado una serie de módulos independientes que sirven para componer automáticamente una canción, y que estará abierta a futuras mejoras o servirá de inspiración para quien venga después.

Nuestra solución está compuesta por dos módulos generadores, uno para la música y otro para las letras, y un módulo separado encargado de fusionarlas y evaluar el resultado. Sin embargo, debido a algunas complicaciones durante el proceso de documentación, el módulo de generación de música desarrollado por Guillermo Villeta Torres no se explica en este documento y será descrito por separado en un futuro TFG.

7.1. Generador de letras

Como resultado de nuestro trabajo en este módulo, hemos conseguido la creación de una letra de una canción, en versos, un análisis sintáctico que muestra coherencia entre sujeto y predicado, y un analizador de sentimiento que aporta homogeneidad semántica a lo largo de la letra. Aún con esto, existen muchos puntos de mejora, ya sea en cada módulo o en el proceso general seguido, que desarrollaremos a continuación.

Analizador sintáctico Por un lado, hemos conseguido que el módulo de *análisis sintáctico* verifique la coherencia comprobando el número de sujetos y verbos en la frase.

Para empezar, este enfoque está bien encaminado, sin embargo, podría mejorarse a través del uso, o el desarrollo si fuera necesario, de mejores analizadores de texto que permitan identificar su estructura más eficazmente. Una vez hecho el análisis, podrían preestablecerse algunas reglas que permitiesen comprobar que el texto creado está correctamente estructurado. Entre estas reglas, incluir algunas que permitan la introducción de figuras literarias que alteren el orden del texto, como el hipérbaton, permitiría darle un mayor nivel de poeticidad al texto. Otra posibilidad, sería la introducción de redes neuronales que, añadido a unas reglas predefinidas, permita aprender ciertas figuras e incorporarlas a los textos.

En caso de que este módulo funcionara correctamente, podría utilizarse como herramienta para apoyar a los profesores y alumnos en el análisis sintáctico de las oraciones en sus clases.

Rima y estructura Por otro lado, no nos centramos en conseguir una *rima* perfecta ya que en las letras no tiene tanto peso como en los poemas. Aparte de eso, nuestro enfoque para estructurar la canción ha sido bastante exitoso al ser fácil de modificar al gusto del desarrollador pudiendo establecer la estructura que más se adecue a lo que necesite. Para mejorar la rima, se puede modificar la generación por Cadenas de Markov de las oraciones, incluyendo una condición de que la última palabra debe rimar con la última palabra del primer verso escogido. También se puede incluir ciertos patrones de rima (ABBA, ABAB) en función de cómo queramos que rime el texto. Todo esto serviría como ayuda a escritores y artistas, o como una manera más entretenida de aprender sobre poesía para los alumnos.

Análisis de sentimiento En cuento al *análisis de sentimiento*, el hecho de crear el texto por frases y no en un bucle "infinito" nos ha permitido tener más control en el análisis de sentimientos, ya que estamos comprobando el sentimiento en cada frase y no en el texto general. Para mejorar esta parte, creemos que la importancia no reside en la forma en que se evalúa el texto, sino en la propia evaluación. Esto significa que el evaluador sea capaz de comprobar si una palabra o frase en un contexto expresa felicidad o tristeza en comparación con otro contexto. Además, como existen muchos sentimientos y emociones diferentes, tratar de desarrollar esta rama para conseguir un "analista de emociones" (alegre, triste, enfadado, divertido...) podría mejorar la relación entre una música y su letra. Así mismo, podría ayudar a identificar la letra de un grupo o cantante particular con un género musical. Esto podría realizare mediante el entrenamiento de una red neuronal al que se le especifique la emoción o las emociones de una letra de una canción, y este las analice hasta poder identificar distintas emociones. Además, este análisis podría llevarse al punto de poder identificar o asociar una letra de una canción con un grupo, cantante o género en particular. En este caso, a la red neuronal se le enseñaría con agrupaciones de canciones por grupo, cantante o género, respectivamente. Esto además, podría utilizarse para ayudar a los ya existentes "identificadores de canciones" o a las "aplicaciones de reproducción de música" para encontrar artistas y canciones similares que gusten al oyente y para que le muestren nuevos descubrimientos.

Otros idiomas Por último, desarrollar este módulo de generador de letras de canciones en *otros idiomas* aportaría diferentes perspectivas y enriquecería el proyecto en general. En este caso, la parte más afectada sería el análisis sintáctico debido a las diferencias existentes entre los idiomas provenientes de distintas raíces, como el español, el inglés y el chino.

7.2. Evaluador

En el campo de la creatividad computacional, la composición automática de canciones combinando melodía y letras generadas independientemente es algo que apenas se ha considerado. Hemos visto que ha habido varios enfoques para desarrollar un compositor de música y un generador de poesía usando diferentes técnicas, pero el trabajo de unir estos dos siempre ha sido hecho por una persona. Inicialmente, esto fue un inconveniente debido a la originalidad de la idea y a la falta de trabajo previo en el que basar el desarrollo, pero también deja un amplio margen de mejora.

En la parte de evaluación, hemos logrado desarrollar una aplicación capaz de fusionar un determinado conjunto de música y letras y detectar si encajan o no. Esto es el resultado de un proceso iterativo en el que el análisis de sentimientos de ambas partes juega un papel importante a la hora de determinar el grado de calidad de la fusión de ambos. Junto con este análisis, tanto la música como la letra se estudian independientemente para extraer distinta información relevante que nos ayudará a modificar el nivel de calidad de la canción resultante.

El proceso seguido (descrito en la sección 4.3) trata de imitar el que toma una persona cuando compone una canción. Como es uno de los primeros desarrollos llevados a cabo en este campo, podría ser comparable a un compositor aficionado, que sigue un conjunto de reglas simples y muy teóricas para determinar si la canción resultante es lo suficientemente buena.

Aún así, como se ha expuesto a lo largo de la fase de desarrollo, hay algunas formas en que el evaluador puede ser mejorado.

Utilizar inteligencia artificial o técnicas de aprendizaje profundo Aunque pensamos que las pruebas llevadas a cabo son lo suficientemente significativas, usando un enfoque no iterativo basado en inteligencia artificial o aprendizaje profundo para determinar si la letra se ajusta a la melodía, la calidad de los resultados probablemente mejoraría. Una posibilidad sería utilizar las canciones de un grupo como conjuntos de datos para los generadores de música y poesía, así como para el evaluador. De esta manera, si se implementa correctamente, los resultados de la aplicación serían canciones que sonarían muy similares a las ya compuestas por el grupo elegido o posibles versiones de diferentes canciones, usando técnicas de composición sacadas de otras canciones (o incluso distintos artistas).

Mejorar el estudio de tendencias El enfoque seguido en esta primera versión del evaluador para el estudio de las tendencias en la música para determinar el análisis de sentimientos, sólo considera dos notas consecutivas. Aún así, para conseguir un resultado más sensato, sería interesante que esta prueba se llevara un paso más allá, haciendo grupos más grandes de notas para obtener un estudio más rico de las tendencias. Otra posible manera de enriquecer este estudio de tendencias sería considerar el grado de salto entre dos notas, dando más importancia a notas más distantes y viceversa.

Definir un criterio más completo para la concordancia de sílabas y notas Echando un vistazo a muchas canciones populares y más complejas, es fácil encontrar casos en los que hay varias notas o acordes al principio de la composición que se tocan solos, sin letras cantadas junto a ellos. Tener un sistema que pueda determinar cuál es el mejor momento en la melodía para que comiencen las letras sería muy valioso y significaría un enorme paso hacia la "humanización" del proceso de evaluación y fusión. Además, si después de aplicar este criterio se pudiera decidir si una sílaba debe ser cantada durante más de una nota diferente, los resultados obtenidos verían una gran mejoría. Esto se podría hacer usando técnicas de machine learning o inteligencia artificial, ya que entrenar el módulo con diferentes canciones puede ayudar a determinar el punto óptimo para comenzar y las palabras que podrían alargarse.

Mejorar el cómputo de la puntuación Como se ha visto en las secciones 4.3.6 y 5.2.4, se pueden añadir o mejorar algunos criterios utilizados para el cálculo de los índices de calidad. Comparar la longitud de la sílaba, preferiblemente fonéticamente, con la longitud de la nota ayudaría a lograr una puntuación más determinante. También sería valioso utilizar esta longitud de sílaba para asignarla a varios tonos de nota, como se mencionó en el punto anterior.

Expandir la información del feedback Si se hace junto con los generadores de música y poesía, sería interesante que el evaluador les enviara una información del resultado más completa. Entre la información que podría ser valiosa para estos módulos, si fuera posible determinar si un verso es malo debido a las sílabas acentuadas, la distribución de las palabras o el significado general y el generador pudiera usar esta información, los resultados podrían ver una gran mejoría. Análogamente, poder comunicar al generador de música que una medida no es buena porque su tendencia, integridad o puntuación de análisis de sentimientos, llevaría a la aplicación a mejores resultados y probablemente disminuiría el número de iteraciones necesarias para llegar a él.

Decidir qué módulo tiene más posibilidades de mejorar el resultado Al igual que en el punto anterior, determinar qué modificación del generador proporcionaría un mejor resultado después de la retroalimentación significaría una gran mejora. Esta decisión se podría tomar considerando la puntuación más baja de todas, siempre y cuando estén normalizadas y sean comparables. Al poder hacer esto, el tiempo de ejecución se reduciría significativamente ya que sólo un módulo tendría que rehacer su trabajo y también el resultado probablemente sería mejor después de menos iteraciones.

Chapter 8

Appendix

All the code can be consulted in the following GitHub repository.

`https://github.com/NILGroup/TFG-1819-PoesiaMusica/tree/master/C%C3%B3digo`

Bibliography

If I have seen further than others, it is by standing upon the shoulders of giants.

Isaac Newton

Aamodt, A. and Plaza, E. Case-based reasoning; foundational issues, methodological variations, and system approaches. *AI COMMUNICATIONS*, Vol. 7(1), 39–59, 1994.

Alfonseca, M., Cebrián, M. and De la Puente, A. A fitness function for computer-generated music using genetic algorithms. *WSEAS Transactions on Information Science and Applications*, Vol. 3, 518–525, 2006.

Bailey, R. W. Computer-assisted poetry: the writing machine is for everybody. In *Proceedings of the ICCBR-01 Workshop on Creative Systems*, 283–295. Mitchell, J. L., editor, Computers in the Humanities. Edinburgh University Press, Edinburgh, UK, 1974.

Barbieri, G., Pachet, F., Roy, P. and Degli Esposti, M. Markov constraints for generating lyrics with style. In *Ecai*, Vol. 242, 115–120. 2012.

Charnley, J. W., Colton, S. and Llano, M. T. The flowr framework: Automated flowchart construction, optimisation and alteration for creative systems. In *ICCC*, 315–323. 2014.

Colton, S., Goodwin, J. and Veale, T. Full-face poetry generation. In *ICCC*, 95–102. 2012.

Conklin, D. Music generation from statistical models. *Journal of New Music Research*, Vol. 45, 2003.

Gervás, P. Evolutionary elaboration of daily news as a poetic stanza. In *Proceedings of the IX Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados-MAEB*, 229–238. 2013.

Gervás, P. An expert system for the composition of formal spanish poetry. *Journal of Knowledge-Based Systems*, Vol. 14, 181–188, 2001.

Gervás, P. Computational modelling of poetry generation. *Artificial Intelligence and Poetry - AISB Convention 2013*, 11–16, 2013.

Gonçalo Oliveira, H., Cardoso, A. and Pereira, F. Tra-la-lyrics: An approach to generate text based on rhythm. In *Proceedings of the 4th International Joint Workshop on Computational Creativity*, 47–55. 2007.

Gouyon, F., Dixon, S., Pampalk, E. and Widmer, G. Evaluating rhythmic descriptors for musical genre classification. In *Proceedings of the AES 25th International Conference*, 196–204. 2004.

Hayes, B. and Kaun, A. The role of phonological phrasing in sung and chanted verse. *The Linguistic Review*, Vol. 13, 2001.

Herremans, D. and Chew, E. Morpheus: automatic music generation with recurrent pattern constraints and tension profiles. In *2016 IEEE Region 10 Conference (TENCON)*, 282–285. 2016. ISSN 2159-3450.

Levy, R. P. A computational model of poetic creativity with neural network as measure of adaptive fitness. In *Proceedings of the ICCBR-01 Workshop on Creative Systems*. 2001.

Malmi, E., Takala, P., Toivonen, H., Raiko, T. and Gionis, A. Dopelearning: A computational approach to rap lyrics generation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 195–204. ACM, 2016.

Manurung, H. M. *An evolutionary algorithm approach to poetry generation*. PhD thesis, University of Edinburgh, 2003.

Monteith, K., Francisco, V., Martinez, T., Gervás, P. and Ventura, D. Automatic generation of emotionally-targeted soundtracks. *Proceedings of the 2nd International Conference on Computational Creativity, ICCC 2011*, 2012a.

Monteith, K., Martinez, T. and Ventura, D. Automatic generation of music for inducing emotive response. *Proceedings of the International Conference on Computational Creativity, ICCC-10*, 2012b.

Navarro-Cáceres, M., Oliveira, H. G. and Corchado, J. M. Towards an automated composer of popular spanish songs: Integrating a music generator and a song lyrics generator. *XVIII Conferencia de la Asociación Española para la Inteligencia Artificial*, 2018.

Oliveira, H. G. Poetryme: a versatile platform for poetry generation. *Computational Creativity, Concept Invention, and General Intelligence*, Vol. 1, 21, 2012.

Oliveira, H. G. A survey on intelligent poetry generation: Languages, features, techniques, reutilisation and evaluation. In *Proceedings of the 10th International Conference on Natural Language Generation*, 11–20. Association for Computational Linguistics, Santiago de Compostela, Spain, 2017.

Oliveira, H. G. and Alves, A. O. Poetry from concept maps–yet another adaptation of poetryme's flexible architecture. In *Proceedings of 7th International Conference on Computational Creativity, ICCC*. 2016.

Oliveira, H. G., Hervás, R., Díaz, A. and Gervás, P. Multilingual extension and evaluation of a poetry generator. *Natural Language Engineering*, Vol. 23(6), 929–967, 2017.

Pearce, M., Meredith, D. and Wiggins, G. Motivations and methodologies for automation of the compositional process. *Musicae Scientiae*, Vol. 6, 2002.

Pease, A. and Colton, S. On impact and evaluation in computational creativity: A discussion of the turing test and an alternative proposal. In *Proceedings of the AISB symposium on AI and Philosophy*, Vol. 39. 2011.

Phon-Amnuaisuk, S., Hui Hean Law, E. and Ho, C. Evolving music generation with som-fitness genetic programming. In *Workshops on Applications of Evolutionary Computation*, 557–566. Springer, 2007.

Potash, P., Romanov, A. and Rumshisky, A. Ghostwriter: Using an lstm for automatic rap lyric generation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1919–1924. 2015.

Toivanen, J. M., Toivonen, H., Valitutti, A. and Gross, O. Corpus-based generation of content and form in poetry. In *ICCC*. 2012.

Wong, M. T., Chun, A. H. W., Li, Q., Chen, S. and Xu, A. Automatic haiku generation using vsm. In *WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering*, 7. World Scientific and Engineering Academy and Society, 2008.

Yan, R. i, poet: Automatic poetry composition through recurrent neural networks with iterative polishing schema. In *IJCAI*, 2238–2244. 2016.

Yan, R., Jiang, H., Lapata, M., Lin, s.-d., Lv, X. and Li, X. I, poet: Automatic chinese poetry composition through a generative summarization framework under constrained optimization. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2197–2203. 2013.

Zhang, X. and Lapata, M. Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 670–680. 2014.

It's not about how much we lost,
it's about how much we have left.
We're the Avengers.
We gotta finish this.

Avengers: Endgame
Tony Stark